FLORIDA INSTITUTE OF TECHNOLOGY

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of

Master of Science
in
Computer Science

FIGS
A FORTH INTERACTIVE GRAPHICS SYSTEM

Jer-Ming Lee

APPROVED BY:

Thomas O. Hand

Dr. Thomas O. Hand
Associate Professor and Chairman of the
Graduate Computer Science Program

Joel H. Blatt

Dr. Joel H. Blatt
Professor of Physics

G. R. Cuthbert

Ms. G. R. Cuthbert
Adjunct Instructor of
Computer Science

FIGS

A FORTH INTERACTIVE GRAPHICS SYSTEM

by

Jer-Ming Lee

B.S. in C.S., Tamkang University, R.O.C., 1979

Submitted to the Graduate Faculty

in partial fulfillment of

the requirements for the degree of

Master of Science

in

Computer Science

Florida Institute of Technology

1985

Jer-Ming Lee

Jer-Ming Lee

ACKNOWLEDGEMENT

The author wishes to acknowlege his theses advisor, Dr. Thomas O. Hand for his abundant lectures, invaluable advice, great ideas and comments.

The author also wishes to acknowledge his Graphics teacher, Ms. G. R. Cuthbert for her exciting lectures on Interactive Graphics.

The author wishes to thank Dr. Joel H. Blatt for reading this thesis.

The author wishes to dedicate this achievement to his wife Shin-Fang for her constant encouragement and her unending patience in typing.

TRADEMARKS

ATARI 800 XL is a trademark product of Atari, Inc.

KoalaPad Touch Tablet is a trademark product of Koala Technologies Corporation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

The purpose of this thesis is to develop an interactive color graphics environment implemented in the FORTH language on the ATARI 800 XL computer. This environment contains two main parts.

The first part is called the FORTH Interactive Graphics System (FIGS). FIGS provides capabilities to switch graphics modes, change colors, create graphics on the screen surface, save graphics images in disk files, and reload them as needed.

The second part is an application of FIGS called the FORTH Interactive Graphics Editor (FIGE). FIGE allows the user to select graphics commands from the graphics menus and to create graphics on the screen surface by using the KoalaPad Touch Tablet and the ATARI keyboard. It provides a four-colored graphics screen with resolution of 160 by 80. The images on the graphics screen can be scaled by a factor of 2 or 4. The disk operations for saving screen data and reloading screen data are also available in FIGE.

I. INTRODUCTION

The purpose of this thesis is to develop an Interactive Computer Graphics Environment implemented in the FORTH language. This environment contains two main parts. The first part is the kernel of the environment, called the FORTH Interactive Graphics System (FIGS), which provides a collection of primitive graphics commands. The second part is an interactive graphics editor, called the FORTH Interactive Graphics Editor (FIGE), which utilizes these primitive graphics commands. It provides capabilities for creating graphics on a display surface by using the KoalaPad and the ATARI keyboard. Figure I-1 shows the main menu for the FORTH Interactive Graphics Editor. A brief description for each selection is listed below:

Point: Plot a point at the present cursor position.

Line: Draw a line between two points.

Draw: Plot points determined by the trace of the cursor.

Erase: Erase points determined by the trace of the cursor.

Text: Draw a text string which is entered from the keyboard starting at the present cursor position.

Box: Draw a rectangle.

Fbox: Draw a filled rectangle.

Circle: Draw a Circle.

Fcircle: Draw a filled circle.

Fill: Fill a connected closed region with the current color.

Copy: Copy a rectangular region to another place.

Move: Move a rectangular region to another place.

Clear: Clear the whole screen.

Bclear: Clear a rectangular (box-like) region.

Setcolor: Display a color menu.

Disk: Disk operations:

- Show directory
- Save screen data
- Load screen data
- Delete a file
- Format a disk

Printer: Reserved for further implementation.

Zoom: Set zoom mode to either normal, double or quadruple.

Usebrush: Select a brush for either Draw or Erase.

D<-->B: Adjust the luminance of the current color (Darker or Brighter).

Exit: Exit to the FORTH interpreter.

Go: Go ahead and execute the selected commamd.

Usecolor: Select a color number from the available (four) color numbers.

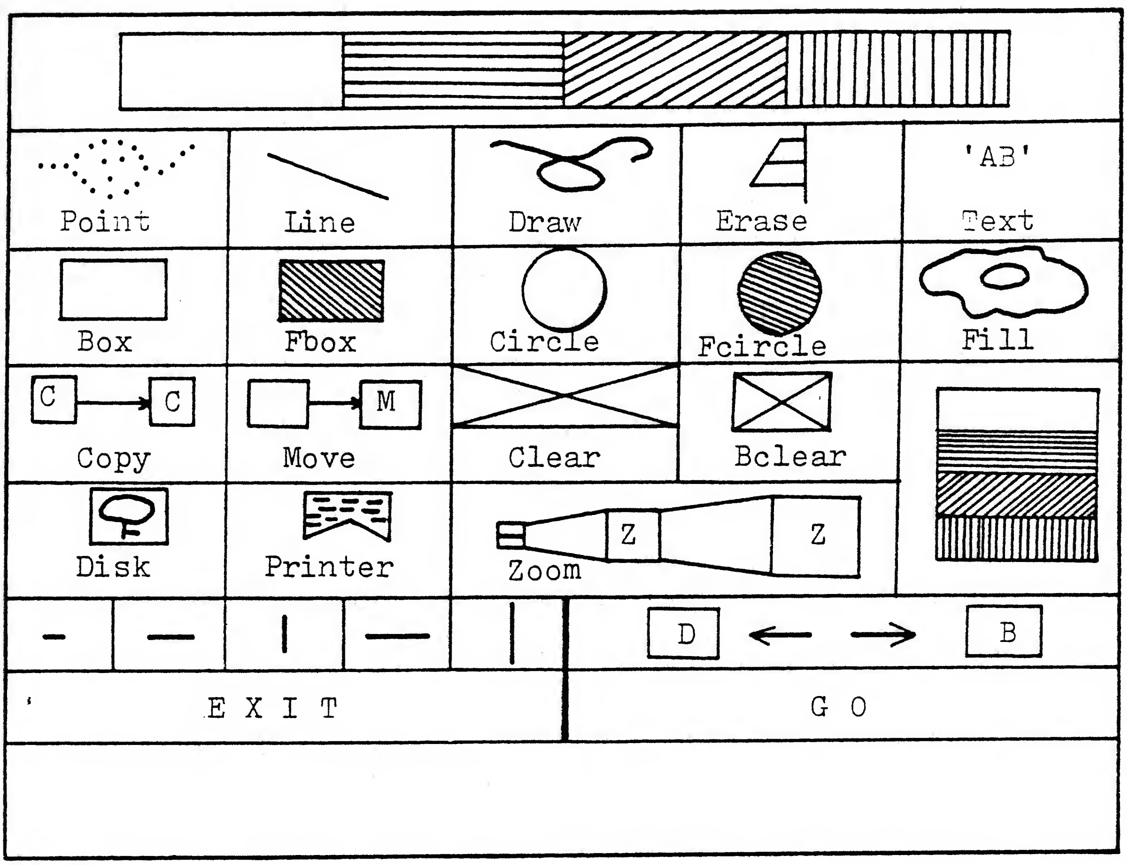


Figure I-1 Main Menu -- FORTH Interactive Graphics Editor

Before getting started with the over-all system design, there are two important questions that must be considered. They are

1. What is Interactive Computer Graphics?
2. Why is FORTH suitable for Interactive Computer Graphics?

According to James D. Foley(1), Interactive Computer Graphics is a form of man-machine interaction in which a user dynamically controls the picture's content, format, size, or colors on a display surface by means of interaction devices such as a keyboard, lever, or joystick. Because of this interaction, the speed becomes a very important factor in designing the system.

FORTH has been successfully used in several applications, such as, real-time system control(2), database management(3), Turtle Graphics(4), and robotics. FORTH has significant features that have made it successful. Some of these features, which are listed below, are beneficial for developing an Interactive Computer Graphics Environment.

1. Interactive Nature

   Since FORTH is an interactive programming environment, it certainly meets the interactive reguirement. Actually, FORTH consists of two interpreters along with several compilers.

2. Faster Speed

   Since FORTH is a threaded language, there is less overhead than required in most other high level languages.

3. Extensibility

   This feature aids with constructing complex graphics images from more elementary images.

4. Selectiveness to Specific Application

   By using FORTH's vocabularies, specific application oriented words can be designed that reduce the command search time at compile time.

5. Ease to Interface to Assembly Language

   By using a FORTH assembler, time critical words can be developed as low-level words that speed up execution. Also, operating system routines can be easily called.

6. Compactness of Code

   This feature simply saves memory which allows the extra memory to be used for other features.

Observing the fitness of FORTH to do interactive graphics, I decided to use it to develop an interactive graphics environment for my thesis.

This environment has been developed on the ATARI 800 XL home computer, because it has several significant graphics features and because it is available in the Computer Science department.

Several preliminary chapters are presented on material necessary for the development of this thesis. A system overview of the ATARI 800 XL computer is presented in chapter II. A memory map for the ATARI fig-FORTH is presented in chapter IV. Since the FORTH Interactive Graphics Editor uses the KoalaPad Touch Tablet as the major pointing device, an introduction to the KoalaPad Touch Tablet is presented in chapter III.

The two main parts of this thesis are presented in chapter V and chapter VI. Only the important words are discussed in these chapters although the complete source listing for the thesis appears in Appendices A and C.

II. ATARI 800 XL Computer System Overview

A. Internal Hardware Components

The internal hardware layout for the ATARI 800 XL computer is very much different from other personal computers. Besides the basic components of the computer system, such as a 6502 microprocessor, RAM, ROM, and a PIA, it also has three special-purpose chips named ANTIC, CTIA, and POKEY. Figure II-1 illustrates the hardware arrangement.

ANTIC is a microprocessor dedicated to the television display. It has its own specialized instruction set. Its programs and data are written in RAM by the 6502. ANTIC executes its programs, called display lists, and retrieves data (display data) from RAM by using direct memory access (DMA).

CTIA is a television interface chip. It converts the digital commands from ANTIC or the 6502 into a signal that goes to the television. It also has some features of its own, such as handling color, player-missile graphics, and collision detection.

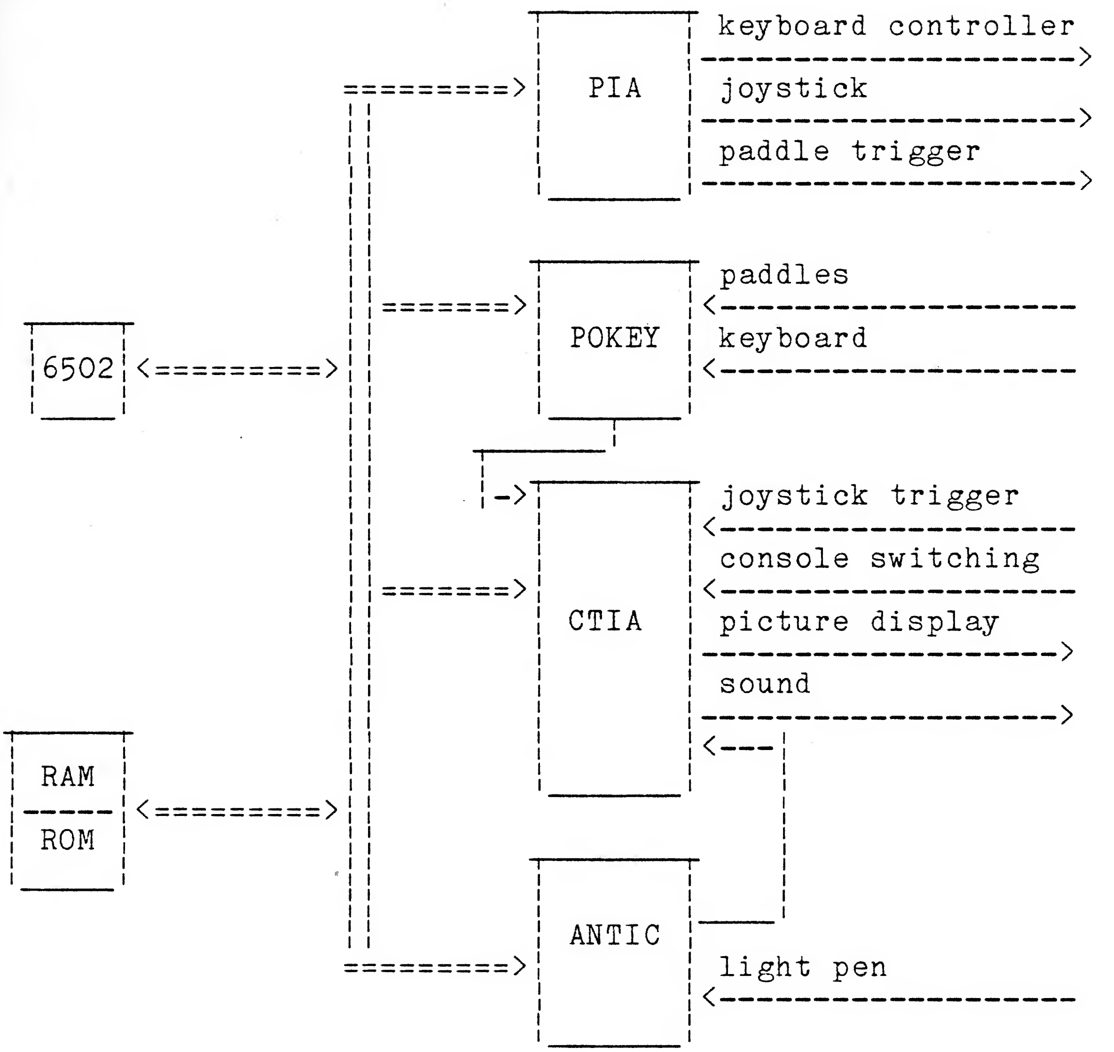


Figure II-1 Hardware Arrangement for the ATARI 800 XL Computer

POKEY is a digital input/output chip. It handles serial I/O bus, audio-generation, keyboard scan, and random number generation. It also digitizes the resistive paddle inputs and controls maskable interrupt requests from peripherals.

All three of these special chips (ANTIC, CTIA, POKEY) function simultaneously with the 6502. They remove a lot of the load from the 6502 and speed up the overall system execution.

B.Graphics Features

1. Text/Graphics Modes

There are 6 text modes and 8 graphics modes available to ANTIC. Some of the modes have 4 colors available while the others have only 2 colors available. Pixel widths and mode line widths vary with the mode. Table II-1 illustrates these details.

ANTIC Mode	BASIC Mode	Nbr of Colors	Scan Lines/ Mode Line	Pixels/ Mode line	Bytes/ line	Bytes/ screen
2	0	2	8	40	40	960
3	none	2	10	40	40	760
4	none	4	8	40	40	960
5	none	4	16	40	40	480
6	1	5	8	20	20	480
7	2	5	16	20	20	240
8	3	4	8	40	10	240
9	4	2	4	80	10	480
A	5	4	4	80	20	960
B	6	2	2	160	20	1920
C	none	2	1	160	20	3840
D	7	4	2	160	40	3840
E	none	4	1	160	40	7680
F	8	2	1	320	40	7680

Table II-1 ANTIC Mode Line Requirements

2. Display Lists

A display list is a program executed by ANTIC. It provides the flexibility for controlling the display. Different graphics modes can be mixed on the same screen. ANTIC's instruction set is rather simple but at the same time is quite powerful. Table II-2 gives the display list instruction set. Besides the four classes of normal instructions, such as graphics mode, text mode, blank line, and jump, there are also four special options, namely, display list interrupt (DLI), load memory scan (LMS), vertical scroll, and horizontal scroll. Figure II-2 illustrates the bit arrangement for the four options of a display list instruction. DLI can be used to add more colors to a display. A load memory scan can be used to direct ANTIC to fetch display data from noncontiguous RAM areas. Vertical scrolling and horizontal scrolling can be used to aid with fine scrolling of the display. These are powerful tools which can be used to enhance the graphics; they will be used in this environment.

Instruction (HEX)	Meaning
0	1 blank line
10	2 blank lines
20	3 blank lines
30	4 blank lines
40	5 blank lines
50	6 blank lines
60	7 blank lines
70	8 blank lines
1	jump to location
41	jump and wait for vertical blank interrupt
2--F	14 display instructions corresponding to 14 ANTIC modes

Table II-2 Display List Instruction Set

```
 7   6   5   4   3   2    1  0
_______________________________
|   |   |   |   |              |  Diplay list instruction
|___|___|___|___|______________|
  |   |   |   |        |
  |   |   |   |        |----->  ANTIC mode number (2--F)
  |   |   |   |
  |   |   |   |-------------->  Vertical scroll
  |   |   |
  |   |   |------------------>  Horizontal scroll
  |   |
  |   |---------------------->  Load memory screen (LMS)
  |
  |-------------------------->  Display list interrupt (DLI)
```

Figure II-2   Four Options for the Display List Instruction

3. Colors

Color values are kept in one-byte color registers. CTIA provides 16 hue values and 8 luminance values for each hue giving a total of 128 possible colors. However, they are restricted both by the number of color registers and by the graphics modes. In particular, only 2 or 4 colors can be displayed in a normal mode line. As has been mentioned previously, additional colors on the same screen can be achieved by using the DLI.

4. Player-missile Graphics

The conventional way to achieve animation in computer graphics is to move the image data through the screen RAM area. However, there are two problems with this technique. First, if the animation is being done in a graphics mode with large pixels, the motion will not be smooth. Second, the screen is a two-dimensional image, but the screen RAM is one-dimensional.

The ATARI computer provides player-missile graphics to solve these problems. Player-missile graphics is separated from normal graphics modes (playfields). There are 4 players and 4 missiles provided by CTIA. Each player or missile has its own color register, horizontal position register, and shape data area in RAM. The horizontal motion of a player-missile is easy to handle by the user.

Simply put the horizontal position value into its horizontal position register. However, the user is responsible for handling all vertical motion of the player-missiles. A simple way to achieve this is to move the player-missiles shape data through its own data area. There are two kinds of vertical resolution (single line and double line), and three kinds of horizontal width (normal, double, and quadruple) available in player-missile graphics. Because player-missiles are separated from playfields, they can be used efficiently as screen cursors. They also can be used to add colors in the same mode line.

III KOALAPAD TOUCH TABLET

The KoalaPad Touch Tablet is a pointing device developed by Koala Technologies Corporation. It has three main features:

1. Pressure Sensitive Drawing Surface

   A point on the surface can be selected by pressing either the stylus provided or your finger at that point. The KoalaPad Touch Tablet's internal circuitry converts pressure on the surface to location information that is sent to the computer.
2. Two Control Buttons

   These buttons may be used to either confirm menu selections or present data entries.
3. Stylus

   This is a pencil-like device specially designed to select a point on the surface by pressing it at that point.

The position of the stylus on the tablet's flat surface is defined by horizontal and vertical coordinates. The horizontal coordinate is read as PADDLE 0 and the vertical coordinate is read as PADDLE 1. Similarly, the two touch tablet buttons are treated as paddle trigger 0 (left button) and paddle trigger 1 (right button). Table

III-1 illustrates the port assignments and read values for the KoalaPad Touch Tablet as used on the ATARI 800 XL computer.

Descriptions	Port name	Memory location (HEX)	Shadowed memory (HEX)	Read values
stylus position				
horizontal	PADDLE 0	D200	270	3-228@
vertical	PADDLE 1	D201	271	3-228@
tablet buttons				
left	PTRIG 0	D300*	27C	0-pressed 1-not pressed
right	PTRIG 1	D300*	27D	0-pressed 1-not pressed

@ 3 is the default value when stytus is not on
* D300 is the memory location for PORTA of the 6502 PIA

Table III-1 Port Assignments and the KoalaPad Touch Tablet Interface

The range of posible values for horizontal and vertical coordinates on the KoalaPad are 3 (the extreme left or top) through 228 (the extreme right or bottom). If the stylus is not pressed against the tablet, the default paddle values are both 3. The possible values for the two touch tablet buttons are either 0 (pressed) or 1 (not pressed).

Unlike joysticks or game paddles, touch tablets can get from point A to point B by skipping all intermediate points. This direct positioning capability makes touch tablets easier to use for selecting menu entries, moving game players and drawing computer graphics. For this reason, the KoalaPad Touch Tablet was chosen as the major pointing device in the FORTH Interactive Graphics Editor.

IV. ATARI Fig-FORTH MEMORY MAP

The memory map for the ATARI fig-FORTH is similiar to the standard fig-FORTH memory map. However, there are some special assignments in ATARI fig-FORTH.

1. The return stack is the same as the system stack (from 1FF down to 100).
2. The parameter stack is allocated on the zero-page RAM (from BC down to 92).
3. Block size is 128 bytes.
4. Disk buffers, user area, terminal input buffer, and stacks are all allocated in memory below the dictionary.

Figure IV-1 illustrates the memory map for the ATARI fig-FORTH environment.

```
       FFFF
                       ____________________
                      |                    |
                      |        ROM         |
       C000 -------> |                    |
                      | ------------------ |
       MEMTOP ----> | screen buffers     |
                      |____________________|

                       ____________________
                      |    text buffer     |
                      |____________________| <------ PAD @
                      | "word" buffer      |
                      | ------------------ |
       DP @ ------->  |                    |
                      |    dictionary      |
                      |                    |
                      | ------------------ |
                      |boot-up literals    |
                      |____________________| <------ ORIGIN (F00)

 (E40) LIMIT ----->  ____________________
                      |   disk buffers     |
 (600) FIRST ----->  |                    |
                      |____________________|

                       ____________________
                      |     user area      |
 (480) ----------->  |____________________|

 (1FF) R0 @ ------->  | return stack    /  |
                      |          _______/  | <------ IN @
       RP @ ------->  |    /  terminal     |
                      |   / input buffer   |
                      | /__________________| <------ TIB @ (100)

  (BC) S0 @ ------->  ____________________
                      | parameter stack    |
       SP @ ------->  |____________________|
                      |      page 0        | <------ 0 PAGE
    0000              |____________________|
```

Fig IV-1   Memory Map for the ATARI Fig-FORTH Environment

V. FORTH INTERACTIVE GRAPHICS SYSTEM

The purpose of the FORTH Interactive Graphics System is to provide a FORTH environment in which users can create graphics on the display surface by using graphics commands. The scope of this chapter is concerned primarily with these graphics commands. Generally, the descriptions for these graphics commands are concerned with three considerations. They are

1. What is the function of the command?
2. How is the command implemented?
3. How is the command used?

A. General Concepts

Suppose that one wants to plot a pixel on the television screen. What must the computer do to accomplish this? What must happen at the interface between the computer and the television set? Since the television set uses what is called a raster beam display system, it does not maintain the image. Consequently, the computer must maintain the screen image in a buffer and consistently send signals to the television telling it what to display. This action requires full-time attention. For this reason, most microcomputers have special designed hardware circuits

to handle it. Figure V-1 shows the conceptual arrangement for the basic system as used by most microcomputer systems.

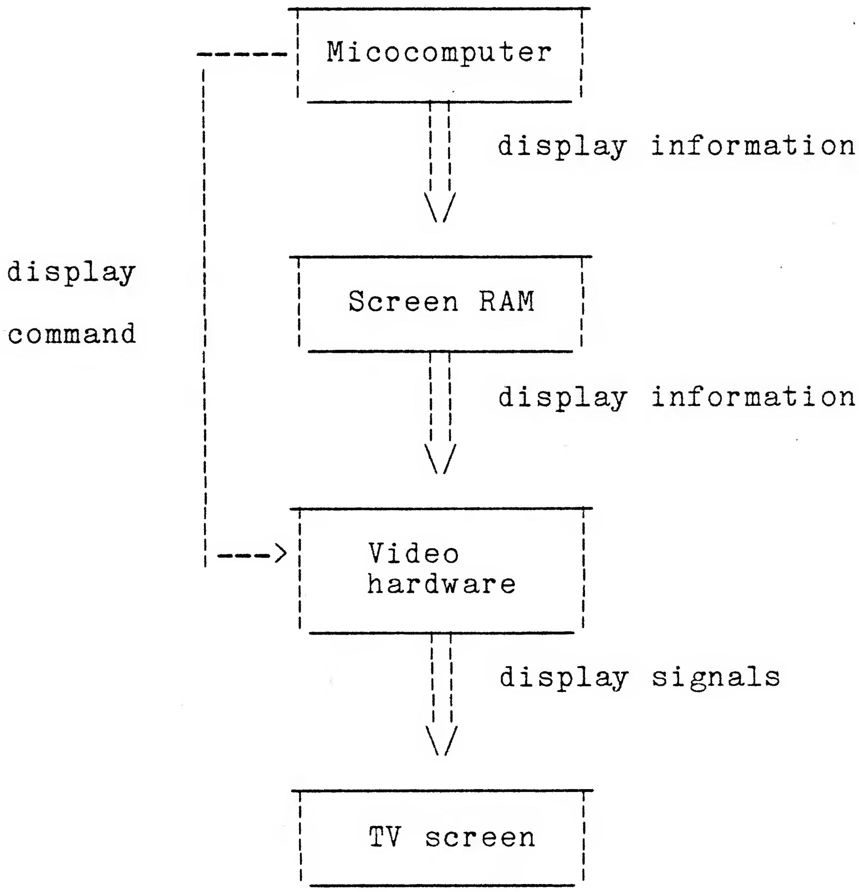


Figure V-1 Conceptual Arrangement of the Display System

This arrangement releases some of the burden from the CPU of the microcomputer.

Fortunately, the ATARI 800 XL microcomputer has two special chips that were designed for the television interface. The ANTIC executes the display lists and converts screen image data into digital commands which are to be sent to the CTIA. The CTIA, then, converts the digital command's into signals for the television.

The programmer prepares the display lists and the screen image data and sets or resets the appropriate registers to control this special hardware. Consequently, the following questions become important where the graphics system is concerned:

1. How are display lists created?
2. How is memory for display lists and screen image data area allocated?
3. How are two-dimensional coordinates transformed into memory locations?
4. How are out-of-boundary conditions that potentially could destroy other useful data managed?
5. How are the coordinates for certain graphics shapes, such as line, circle and boxes calculated?
6. How are pixel data placed into screen image data areas?

7. How are colors set and used?

Once the solutions of these questions have been presented and graphical pictures can be created, it will be necessary to both save the pictures on disk and load them back into memory at a later time.

B. System Overview

The FORTH Interactive Graphics System is designed as an environment which is an extension of the ATARI fig-FORTH System. It acts as an interface between ATARI FORTH users and the graphics display processors (ANTIC, CTIA). Figure V-2 shows the relationship between them. It provides groups of commands which can be used to switch graphics modes, to change colors, to draw figures in the screen image RAM and to save display lists and/or screen image data on disk. The basic system approach is based upon the general concepts which were stated in the last section. Furthermore, the algorithms implemented in this thesis answer the questions asked in the last section.

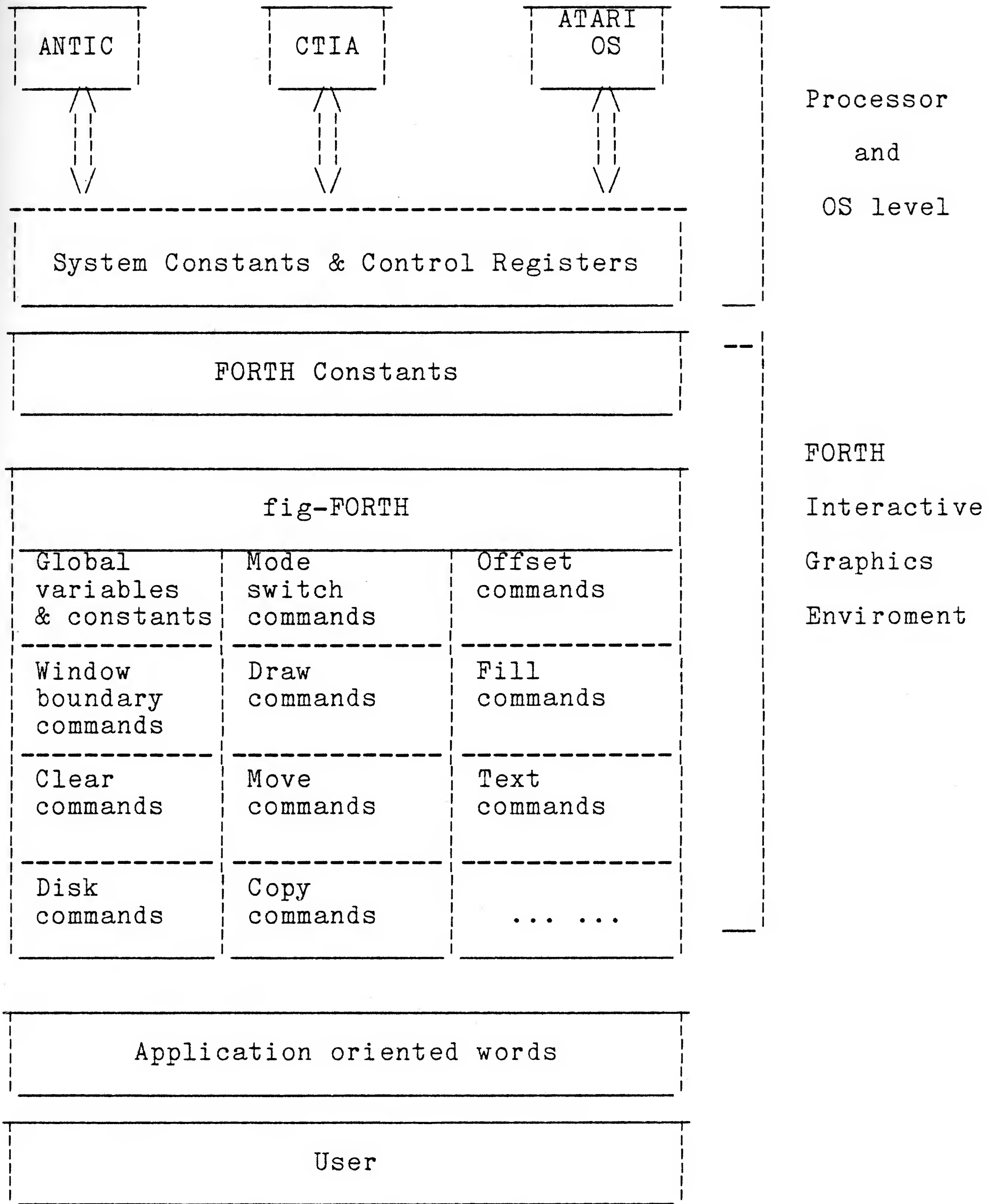


Figure V-2 Conceptual System Layout of FIGS

FIGS can be subdivided into several categories of commands by the function that they perform. These are

1. Mode switching commands.
2. Offset manipulation commands.
3. Window boundary manipulation commands.
4. Color manipulation commands.
5. Screen cursor manipulation commands.
6. Plot point commands.
7. Draw line commands.
8. Draw box commands.
9. Draw circle commands.
10. Draw text string commands.
11. Fill commands.
12. Clear screen commands.
13. Box region moving or copying commands.
14. Scale or zoom commands.
15. Player cursor manipulation commands.
16. Disk file management commands.
17. KoalaPad interfacing commands.
18. Display list interrupt handling commands.

The remaining sections of this chapter present detail descriptions for each of these categories of commands.

C. Detail Description of Commands

In this section, more detailed descriptions of the commands are presented.

1. Mode Switching Commands

These commands create display lists for specific graphics modes. They can also set default values for the environment, such as color values, text window boundaries, number of scroll lines, screen boundaries, and initial cursor position. Here is the general psuedocode for the mode switching commands.

INPUT: none

OUTPUT: none

```
PROCEDURE   Mode switching commands
   Create a display list in high portion of RAM
   Set default values
   Turn off the screen display
   Store the starting address of created display list
             in SDLSTL (location 230 H)
   Turn on the screen display
ENDPROC
```

To create a display list, there are three main words that can be used, namely BEGIN-DL, MODE-LINE and END-DL. Since all nine of the graphics modes provided by ATARI BASIC begin with 24 blank scan-lines (which equals 3 8-blank mode-lines), BEGIN-DL puts 3 bytes of 70H into memory, starting at the address that is on top of the parameter stack. MODE-LINE creates that part of the display list that begins with a load memory scan option instruction. END-DL concludes the display list with a jump instruction that directs ANTIC to fetch the display list instructions starting at the begining of the display list. Please refer to Table II-2 for more information about the display list instructions.

a. SCREEN-OFF

SCREEN-OFF turns off ANTIC by storing 0 in SDMCTL (location 22FH). It also keeps the old value of this control register in a temporary location for possible restoring by the SCREEN-ON command.

b. SCREEN-ON

SCREEN-ON turns on ANTIC and restores other DMA functions that had been previously working before the SCREEN-OFF command turned them off.

The next nine commands are used to create mode environments which correspond to those provided by ATARI BASIC. Refer to Table II-1 for more information.

c. MODE0

This command switches the present graphics mode to graphics mode 0. See the start of this section for the appropriate pseudocode.

d. MODE1

This command switches the present graphics mode to graphics mode 1. See the start of this section for the appropriate pseudocode.

e. MODE2

This command switches the present graphics mode to graphics mode 2. See the start of this section for the appropriate pseudocode.

f. MODE3

This command switches the present graphics mode to graphics mode 3. See the start of this section for the appropriate pseudocode.

g. MODE4

This command switches the present graphics mode to graphics mode 4. See the start of this section for the appropriate pseudocode.

h. MODE5

This command switches the present graphics mode to graphics mode 5. See the start of this section for the appropriate pseudocode.

i. MODE6

This command switches the present graphics mode to graphics mode 6. See the start of this section for the appropriate pseudocode.

j. MODE7

This command switches the present graphics mode to graphics mode 7. See the start of this section for the appropriate pseudocode.

k. MODE8

This command switches the present graphics mode to graphics mode 8. See the start of this section for the appropriate pseudocode.

2. Offset Manipulation Commands

These commands manipulate offsets within the screen. They provide a redraw capability; the same picture can be redrawn at a different position by simply changing the offset values and by executing the same graphics command again. The main idea is to maintain a pair of X-Y offset values, called the x-offset and the y-offset. These values are to be added to the normal screen coordinates during the excution of the graphics draw commands. The x-offset is maintained in the FORTH variable XO while the y-offset is maintained in the FORTH variable YO.

a. OFFSET!

This command stores new offset values in the FORTH variables XO and YO.

INPUT: X-offset
       Y-offset
OUTPUT: none

```
PROCEDURE  OFFSET!
   Store the y-offset in the variable YO
   Store the x-offset in the variable XO
ENDPROC
```

b. OFFSET@

This command puts the current offset values on top of the parameter stack. It provides a convenient way for the user to check the current offset values.

INPUT: none

OUTPUT: X-offset

Y-offset

```
PROCEDURE   OFFSET@
   Fetch the x-offset from the variable XO
   Fetch the y-offset from the variable YO
ENDPROC
```

c. +OFFSET

This command transforms the normal X-Y coordinate values into the offset coordinate values by adding the corresponding offset values to the X-Y coordinate values. It is called during the execution of the graphics draw commands.

INPUT: X -- X-coordinate before adding the x-offset value
Y -- Y-coordinate before adding the y-offset value

OUTPUT: X-coordinate after adding the x-offset value
Y-coordinate after adding the y-offset value

```
PROCEDURE   +OFFSET
   Add the x-offset to X
   Add the y-offset to Y
ENDPROC
```

d. OFFSET>T

This command temporarily saves the current offset values.

INPUT: none

OUTPUT: none

```
PROCEDURE   OFFSET>T
   Move the x-offset to the temporary variable TXO
   Move the y-offset to the temporary variable TYO
ENDPROC
```

e. T>OFFSET

This command restores the current offset values to the values previously saved in TXO and TYO.

INPUT: none

OUTPUT: none

```
PROCEDURE   T>OFFSET
   Move  the x-offset in the temporary variable TXO to  the
              current offset variable
   Move  the y-offset in the temporary variable TYO to  the
              current offset variable
ENDPROC
```

3. Window Boundary Manipulation Commands

These commands are used to set or check window boundary conditions. The main purpose is to prevent data, which is in critical memory areas, from being destroyed by attempting to draw something outside of the proper screen boundaries. The screen boundaries are kept in two sets of variables. The first set consists of TXMIN, TYMIN, TXMAX, and TYMAX. They are used to keep the terminal coordinate boundaries which varies from mode to mode. The second set consists of XMIN, YMIN, XMAX, and YMAX. They are used to keep the logical window boundaries.

Normally, once a certain graphics mode is selected and used, the contents of the first set of variables are saved. This allows the logical window boundaries to be restored to the whole terminal screen. On the other hand, the logical window boundaries can be dynamically changed at will.

a. TWINDOW

This command sets the current terminal screen boundaries.

INPUT: XMIN -- low limit of the X-coordinate
YMIN -- low limit of the Y-coordinate
XMAX -- high limit of the X-coordinate
YMAX -- high limit of the Y-coordinate
OUTPUT: none

```
PROCEDURE  TWINDOW
   Store  high limit of  the Y-coordinate in  the  variable
             TYMAX
   Store  high limit of  the X-coordinate in  the  variable
             TXMAX
   Store  low  limit of  the Y-coordinate in  the  variable
             TYMIN
   Store  low  limit of  the X-coordinate in  the  variable
             TXMIN
ENDPROC
```

b. WINDOW

This command sets the current logical screen boundaries. It provides the capability to restrict the effects of the graphics draw commands to a specific window. With the aid of the offset commands, it then allows the user to draw within certain restricted regions of the display. The input, output and psuedocode are almost the same as that for TWINDOW except that the second set of variables (XMIN, YMIN, XMAX and YMAX) are used in WINDOW.

c. TW>W

This command restores the current logical window boundaries to the current terminal screen boundaries.

INPUT: none

OUTPUT: none

```
PROCEDURE   TW>W
   Fetch  the low limit of the X-coordinate in the terminal
              screen
   Fetch  the low limit of the Y-coordinate in the terminal
              screen
   Fetch the high limit of the X-coordinate in the terminal
              screen
   Fetch the high limit of the Y-coordinate in the terminal
              screen
   Call  WINDOW  to set the logical window  boundaries  with
              these limits
ENDPROC
```

d. BOUNDARY?

This command checks whether a given pair of X-Y coordinates are within the logical window boundaries. This word is used by each draw command to restrict their effect to within the selected logical window.

```
INPUT:  X -- X-coordinate of the given point
        Y -- Y-coordinate of the given point
OUTPUT: F -- result flag
             1: within the window
             0: outside of the window

PROCEDURE  BOUNDARY?
   IF  the  X-Y coordinates are within the  logical  window
             THEN
      Set the result flag to 1
   ELSE
      Set the result flag to 0
   ENDIF
ENDPROC
```

4. Color Manipulation Commands

These commands select colors and place them in specific color registers, or select a certain color register as the current color register. Color registers are one-byte in length. The high-order nibble can contain 16 hue values. The low-order nibble contains 8 luminance values. Luminance values must be even because bit 0 is ignored by the ATARI 800 XL computer. Figure V-3 illustrates the bit arrangement within a color register.

```
 _________________
| h h h h l l l x |
|_________________|
    |     |   |
    |     |   |------- not used
    |     |
    |     |----------- luminance
    |
    |----------------- hue
```

Figure V-3 Color Registers

For the reason of readability, the color values are defined as FORTH constants with appropriate names. Table V-1 shows some color values and their corresponding names.

Color values (HEX)	Color names
0	BLACK
10	RUST
20	RED-ORANGE
30	DARK-ORANGE
40	RED
50	DARK-LAVENDER
60	COBALT-BLUE
70	ULTRAMARINE-BLUE
80	MEDIUM-BLUE
90	DARK-BLUE
AO	BLUE-GREY
BO	OLIVE-GREEN
DO	DARK-GREEN
EO	ORANGE-GREEN
FO	ORANGE

Table V-1 Color Assignments

The following two commands deal with changing the luminance of a color which is contained in a color register.

a. DARKER

This command decreases the luminance value in the given color register by two so that the color becomes 1 level darker. However, if the value is 0, it is changed to the brightest luminance value of OE (hex).

INPUT: color number

OUTPUT: none

```
PROCEDURE   DARKER
   Convert the color number to a color register location
   Fetch the color value from the given color register
            location
   Duplicate the color value
   Substract 2 from the second color value
   Form a new color value with the high-order nibble of the
            first color value and low-order nibble of the
            second color value
   Store the new color value in the given color register
ENDPROC
```

b. BRIGHTER

This command increases the luminance value in the given color register by two so that the color becomes 1 level brighter. However, if the value is OE (hex), it is changed to the darkest luminance value of O.

The input, output and pseudocode for BRIGHTER are similar to that of the word DARKER except that the word BRIGHTER adds 2 to the color value instead of substracting 2 from the color value.

c. USECOLOR

This command selects one of the four color numbers as the current color number. It also sets the color mask byte (COLORMSK) by taking the color mask value from the color mask table (COLMSKTB). Table V-2 shows the color mask values that correspond to the number of available colors and the color number.

Number of colors available	Color number	Color mask (binary)	Color mask (hex)
2	0	00000000	00
	1	11111111	FF
4	0	00000000	00
	1	01010101	55
	2	10101010	AA
	3	11111111	FF

Table V-2 Color Masks

INPUT: color number

OUTPUT: none

```
PROCEDURE  USECOLOR
   Store the color number in the current color number
         variable
   Get the color mask value from the color mask table
   Store the color mask value in the color mask byte
ENDPROC
```

5. Screen Cursor Manipulation Commands

These commands change the cursor position and read the current coordinates of the cursor. In FIGS, the current cursor column and row coordinates are kept in the variables OLDCOL and OLDROW, respectively. The two cursor manipulation commands simply deal with either storing the given coordinates in the current cursor coordinate variables or fetching the coordinates from these variables.

a. LOCATE

This command moves the cursor position to the position given by the coordinates that are on top of the parameter stack.

```
INPUT:  X-coordinate
        Y-coordinate
OUTPUT: none

PROCEDURE   LOCATE
   Store the given Y-coordinate in the corresponding
            variable
   Store the given X-coordinate in the corresponding
            variable
ENDPROC
```

b. READXY

This command fetches the coordinates of the current cursor position and puts them on top of the parameter stack. This provides the capability of examining the current cursor position.

INPUT: none

OUTPUT: X-corrdinate of the current cursor position

Y-coordinate of the current cursor position

```
PROCEDURE   READXY
    Fetch the X-coordinate of the current cursor position
    Fetch the Y-coordinate of the current cursor position
ENDPROC
```

6. Plot Point Commands

These commands are used to plot points on the display. The main idea is to put the current color number as the pixel value at the proper location in the screen image data RAM.

a. PLOT

This is a general PLOT command which can be used to put a pixel value at the proper location in the screen image data RAM. It may be applied in various graphic modes even if they have different pixel sizes and different resolutions.

In order to plot a pixel value at the proper location, the transformation from a two-dimensional X-Y coordinate to one-dimensional memory location is necessary. The command POSITION does this work.

To transfer from an X-Y coordinate to a memory location, the following formulas are used.

Memory Location = Screen memory + Y * bytes-per-line
+ INT (X / pixels-per-byte)

Pixel number in byte = MOD (X / pixels-per-byte)

where

X and Y are the given coordinates,

screen memory is the start location of screen image RAM,

bytes-per-line is the number of bytes used to hold pixel values of one mode-line,

pixels-per-byte is the number of pixels which one byte can hold.

Since most of the graphics modes, other than the text modes, do not use a whole byte to hold a single pixel value, it becomes impossible to put a single pixel into RAM without being concerned with the other pixels in that same data byte. There is one efficient method of solving this problem. A special byte called the pixel mask is used to mask out the pixel values of the color mask. Table V-3 shows the pixel mask values that correspond to the number of pixels. The complement of the pixel mask is used to mask out the surrounding pixel values of the specific data byte. Consequently, the new pixel value and the unchanged surrounding pixel values can be recombined and stored back in the data byte. The word PXPUT does this work. Figure V-4 shows how this method works with sample values.

Pixels/byte	Pixel position	Pixel mask (binary)	(hex)
1	0	11111111	FF
2	0	11110000	F0
	1	00001111	0F
4	0	11000000	C0
	1	00110000	30
	2	00001100	0C
	3	00000011	03
8	0	10000000	80
	1	01000000	40
	2	00100000	20
	3	00010000	10
	4	00001000	08
	5	00000100	04
	6	00000010	02
	7	00000001	01

Table V-3 Pixel Mask Values

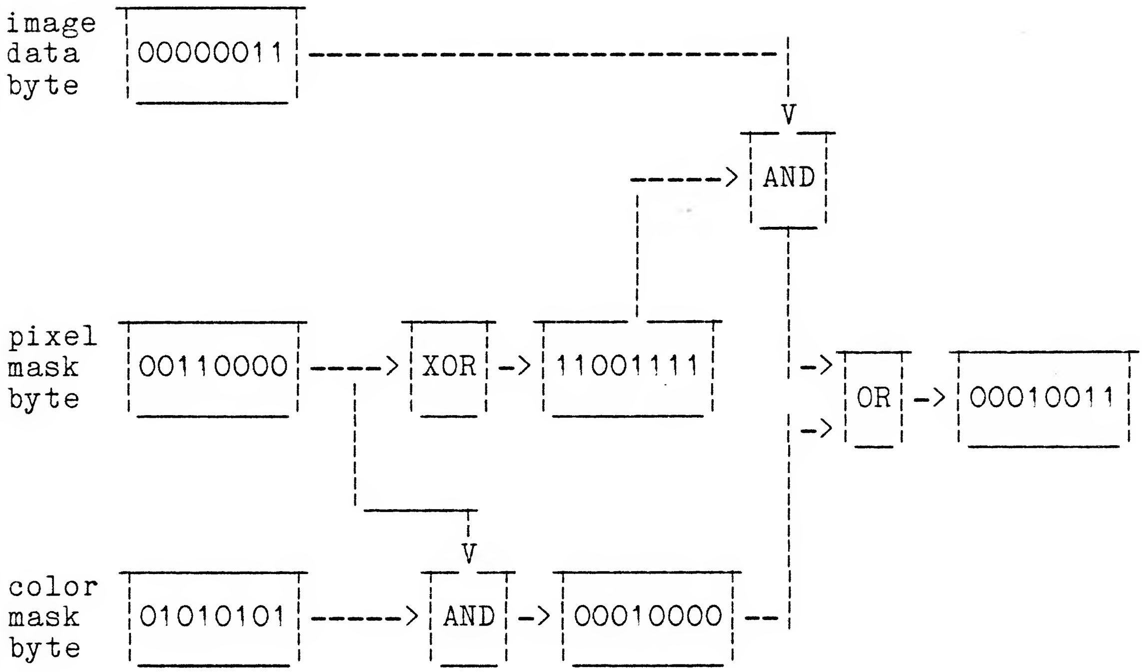


Figure V-4 Storing a Pixel

```
INPUT:  X-coordinate
        Y-coordinate
OUTPUT: none

PROCEDURE   PLOT
   Locate the cursor position with the given coordinates
   Add the offset values to the given coordinates
   Check the boundary
   IF the point is within the boundary THEN
      Transform  the  X-Y coordinates to a memory  location
                and put the pixel number in that byte
      Set the pixel mask byte by using the pixel number
      Put the pixel value into the proper location by using
                the color mask byte and the pixel mask byte
   ELSE
      Do not plot the point
   ENDIF
ENDPROC
```

7. Draw Line Commands

These commands draw line segments on the display from one point to a second point. How can the coordinates of the points that lie on the line segment be calculated?

a. General Line Algorithms

(1). Basic Incremental Algorithms

Let's take a glance at some line drawing algorithms. The simplest approach is to calculate of the slope of the ideal line and use it to increment both the Y-coordinate value and the X-coordinate value to obtain other points on the line. This algorithm is called the Basic Incremental Algorithm(5).

This algorithm requires the usage of floating point arithmetic because slope is a real number. Unfortunately, the floating point arithmetic words provided by ATARI fig-FORTH are not efficient.

(2). Bresenham's Algorithm

Consequently, the Bresenham's Line Algorithm(6) becomes attractive because it requires only integer arithmetic. Figure V-5 illustrates the geometry for Bresenham's algorithm. The black circles are the pixels selected by Bresenham's algorithm.

Bresenham's algorithm uses a decision variable di which at each step is proportional to the difference of s and t. Figure V-5 depicts the ith step, at which the pixel Pi-1 has been determined to be closest to the actual line being drawn, and now it must be decided whether the next pixel to be plotted should be Ti or Si. If s < t, then Si is closer to the desired line and it should be plotted; otherwise, Ti is closer and it should be plotted.

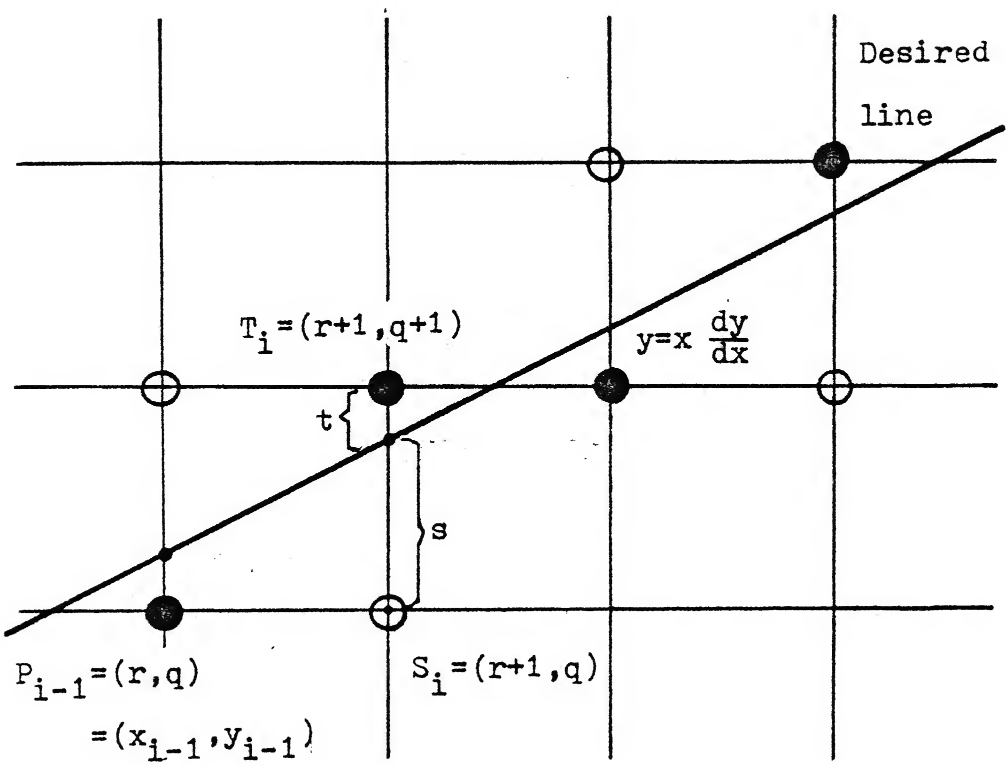


Figure V-5 Bresenham's Algorithm

The derivation is based on the above idea and the resulting equations are listed below:

When $dx > 0$ and $dy > 0$:

If $d_i >= 0$, then

$$d_{i+1} = d_i + 2(dy - dx)$$

else

$$d_{i+1} = d_i + 2dy$$

and

$$d_1 = 2dy - dx$$

However, this version works only for lines with slope between 0 and 1. A general version is derived from a variant of Figure V-5 , and the resulting equations are list below:

When $dx < 0$ and $dy < 0$:

If $d_i < 0$, then

$$d_{i-1} = d_i - 2dy$$

else

$$d_{i-1} = d_i - 2(dy - dx)$$

and

$$d_1 = -(2dy - dx)$$

When $dx > 0$ and $dy < 0$:

If $d_i < 0$, then

$$d_{i+1} = d_i = 2 (dy + dx)$$

else

$$d_{i+1} = d_i + 2dy$$

and

$$d_1 = 2dy + dx$$

When $dx < 0$ and $dy > 0$:

If $d_i < 0$, then

$$d_{i-1} = d_i - 2 (dy + dx)$$

else

$$d_{i-1} = d_i - 2dy$$

and

$$d_1 = - (2dy + dx)$$

The incremental values Inc1 and Inc2 and the initial value for d1 are derived from the equations listed below:

$$Inc1 = (xsign) 2dy$$

$$Inc2 = (xsign) 2 (dy - (xysign) dx)$$

$$d1 = (xsign) (2dy - (xysign) dy)$$

where

$$xsign = \begin{cases} 1, & \text{if } X2 - X1 > 0 \\ 0, & \text{if } X2 - X1 = 0 \\ -1, & \text{if } X2 - X1 < 0 \end{cases}$$

$$\text{ysign} = \begin{cases} 1, & \text{if } Y2 - Y1 > 0 \\ 0, & \text{if } Y2 - Y1 = 0 \\ -1, & \text{if } Y2 - Y1 < 0 \end{cases}$$

$$\text{xysign} = (\text{ xsign }) (\text{ ysign })$$

However, this general version works only for lines with slope between -1 and 1. A few modifications to this algorithm will give a version that handles lines with slopes not between -1 and 1. Simply exchange the x and y coordinates of each point before performing the calculation and exchange them back before drawing the point. The algorithm given below is a general version of Bresenham's Line Algorithm.

```
INPUT:  X1 -- X-coordinate of the start point
        Y1 -- Y-coordinate of the start point
        X2 -- X-coordinate of the end point
        Y2 -- Y-coordinate of the end point
OUTPUT: none

PROCEDURE   Bresenham's Line Algorithm
   Set offset to (X1,Y1)
   Compute dx = |X2 - X1|
   Compute dy = |Y2 - Y1|
```

```
IF dy > dx THEN
   Exchange  X1  and  Y1
   Exchange  X2  and  Y2
   Exchange  dx  and  dy
   Exchange  xsign  and  ysign
   Set  exchanged  indicator  to  1
ENDIF
Compute   Inc1  =  ( xsign ) 2dy
Compute   Inc2  =  ( xsign ) 2 ( dy - ( xysign ) dx )
Compute      d  =  ( xsign ) ( 2dy - ( xysign ) dx )
Set  y to 0
FOR  each  x  FROM  0  TO  ( X2 - X1 )
   IF ( xysign ) d < 0 THEN
      Add  Inc1  to  d
   ELSE
      Add  ysign  to  y
      Add  Inc2  to  d
   ENDIF
   IF exchanged indicator = 1  THEN
      Draw point (y,x)
   ELSE
      Draw point (x,y)
   ENDIF
ENDFOR
ENDPROC
```

b. Special Line Algorithms

Since Bresenham's algorithm is a general line algorithm, it has overheads for some special lines such as horizontal lines and vertical lines.

(1). Horizontal Line

Actually, the data for a horizontal line lies in contiguous bytes of memory. So, to draw a horizontal line, simply fill these contiguous bytes with the value of the color mask and take care of the two bytes that correspond to the ends of the line.

INPUT: X1 -- X-coordinate of the start point

Y1 -- Y-coordinate of the start point

X2 -- X-coordinate of the end point

Y2 -- Y-coordinate of the end point

OUTPUT: none

PROCEDURE Horizontal line

Transform (X1,Y1) into the starting location and the pixel number

Transform (X2,Y2) into the end location and the pixel number

Put the pixel values in the first byte

Fill in the bytes between the starting location and the end location with the value from the color mask

Put the pixel values in the last byte

ENDPROC

(2). Vertical Line

The data for a vertical line have the same pixel number in those bytes which correspond to the points on the line.

INPUT: X1 -- X-coordinate of the start point

Y1 -- Y-coordinate of the start point

X2 -- X-coordinate of the end point

Y2 -- Y-coordinate of the end point

OUTPUT: none

```
PROCEDURE   Vertical line
   Transform  (X1,Y1) into the starting location and  pixel
             number
   Set the address to the starting location
   FOR  each y  FROM  Y1  TO  Y2
      Put  the  pixel value for given pixel number  at  the
               address
      IF Y1 < Y2 THEN
         Add the number of bytes per line to the address
      ELSE
         Substract  the number of bytes per line  from  the
                  address
      ENDIF
   ENDFOR
   Put  the pixel value for the given pixel number  at  the
             address
ENDPROC
```

c. DRAWLINE

This command draws a line between two points which are maintained in the FORTH variables OLDCOL, OLDROW, NEWCOL, and NEWROW. It is the kernel for both the LINE command and the DRAWTO command.

For efficiency, the draw line function is divided into three parts; the general Bresenham's line algorithms, the horizontal line algorithm and the vertical line algorithm.

INPUT: none

OUTPUT: none

```
PROCEDURE   DRAWLINE
   Fetch the coordinates of the first point
   Add the offset values to the first point's coordinates
   Compute the outcode value for the first point
   Fetch the coordinates of the second point
   Add offset values to the second point's coordinates
   Compute the outcode value for the second point
   IF the logical-AND of the two outcode values = 0 THEN
      Compute the direction values
      IF ysign = 0 THEN
         Draw the line using the horizontal line algorithm
      ELSE
         IF xsign = 0 THEN
            Draw the line using the vertical line algorithm
         ELSE
            Draw  the  line using the  general  Bresenham's
                       line algorithm
         ENDIF
      ENDIF
   ENDIF
ENDPROC
```

d. LINE

This command calls the command DRAWLINE to draw a line between the two given points whose coordinates are on top of the parameter stack.

INPUT: X1 -- X-coordinate of the start point

Y1 -- Y-coordinate of the start point

X2 -- X-coordinate of the end point

Y2 -- Y-coordinate of the end point

OUTPUT: none

```
PROCEDURE   LINE
   Store the coordinates of the first point
   Store the coordinates of the second point
   Call DRAWLINE to draw the line
   Position the cursor at the second point
ENDPROC
```

e. DRAWTO

This command draws a line from the current cursor position to the point whose coordinates are on top of the parameter stack. DRAWLINE is called by DRAWTO to do this work.

INPUT: X-coordinate of the point to be drawn to
       Y-coordinate of the point to be drawn to
OUTPUT: none

```
PROCEDURE   DRAWTO
   Store the coordinates of destination point
   Call DRAWLINE to draw the line
   Position the cursor at the destination point
ENDPROC
```

8. Draw Box Commands

a. BOX

This command draws a rectangular region determined by the two opposite-corner points of the rectangular region.

INPUT: X1 -- X-coordinate of the first point
Y1 -- Y-coordinate of the first point
X2 -- X-coordinate of the second point
Y2 -- Y-coordinate of the second point
OUTPUT: none

```
PROCEDURE   BOX
   Call LOCATE to move the cursor position to (X1,Y1)
   Call DRAWTO to draw a line from (X1,Y1) to (X1,Y2)
   Call DRAWTO to draw a line from (X1,Y2) to (X2,Y2)
   Call DRAWTO to draw a line from (X2,Y2) to (X2,Y1)
   Call DRAWTO to draw a line from (X2,Y1) to (X1,Y1)
ENDPROC
```

b. FBOX

This command draws a filled rectangular region determined by two opposite-corner points of the rectangular region.

```
INPUT:  X1 -- X-coordinate of the first point
        Y1 -- Y-coordinate of the first point
        X2 -- X-coordinate of the second point
        Y2 -- Y-coordinate of the second point
OUTPUT: none

PROCEDURE   FBOX
   Add the offset values to the  first point's coordinates
   Compute the outcode value for the first point
   Add  the  offset values to the second point's coordinates
   Compute the outcode value for the second point
   IF the logical-AND of the two outcode values = 0 THEN
      Compute the direction values
      Clip the out of range part
      FOR  y  FROM  clipped Y1  to  clipped Y2
         Draw  a  horizontal  line  from (clipped X1, y)  to
                  (clipped X2, y)
      ENDFOR
   ENDIF
   Position the cursor at (X2,Y2)
ENDPROC
```

9. Draw Circle Commands

These commands are used to draw a circle and a filled circle with a given center and a given radius.

There are several very easy but inefficient ways for drawing a circle. For example, consider a circle with center at the origin with radius R. It's equation is given by

$$X^2 + Y^2 = R^2$$

To draw a quarter circle, we can increment x from 0 to R (radius) in unit steps, solving for the y at each step. The second way is to plot $R\cos\theta$ or $R\sin\theta$ by stepping θ from 0 to 90 degrees. Again, these two methods involve floating point arithmetic and special function calculations which are quite time consuming.

Bresenham has developed a incremental circle generator (7) which is more efficent than either of the above two methods. J. Michener(8) derived the following specific algebraic results and the consequent algorithm by applying Bresenham's methodology.

$d1 = 3 - 2R$

If $di < 0$, then

$di+1 = di + 4Xi-1 + 6$

else

$di+1 = di + 4 (Xi+1 - Yi-1) + 10$

To use J. Michener's algorithm, a supporting algorithm called the Eight-way Symmetry Algorithm(9) has to be used. This supporting algorithm reduces the necessary coordinate calculations to only those for X between 0 to $R/\sqrt{2}$ (the point at which x = y). Figure V-6 shows eight symmetical points on a circle.

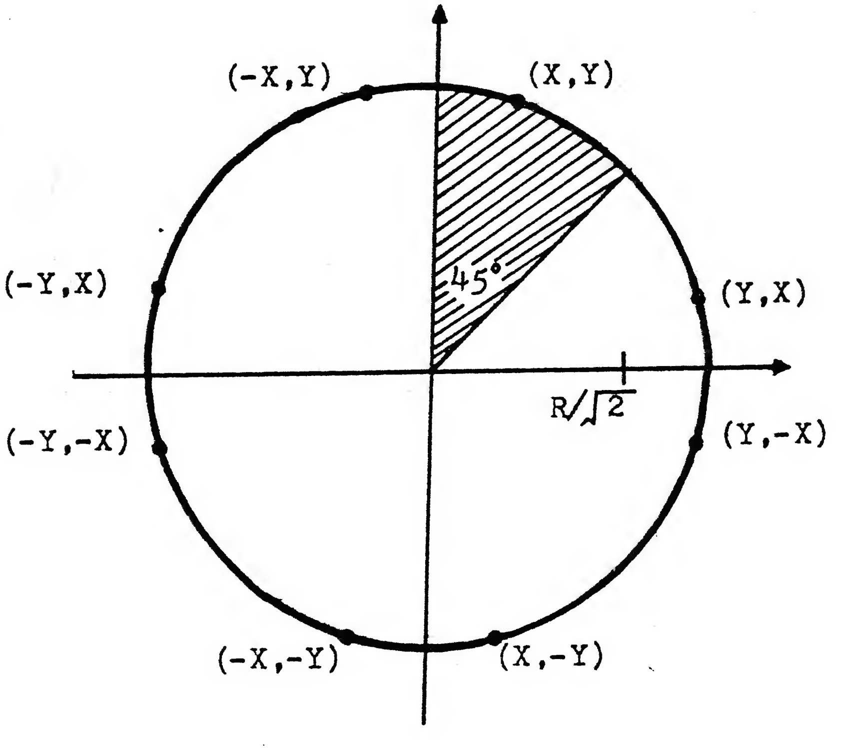


Figure V-6 Eight Symmetrical Points on a Circle

a. CIRCLE-PTS

This command plots the eight symmetric points of the given circle point by applying the Eight-way Symmetry Algorithm. When a circle is to be drawn, CIRCLE-PTS is called by the DRAWCIRCLE command to plot the eight symmetric points for each point on the circle generated by DRAWCIRCLE.

INPUT: X -- X-coordinate of a circle point

Y -- Y-coordinate of a circle point

OUTPUT: none

```
PROCEDURE   CIRCLE-PTS
   Draw the point (x,y)
   Draw the point (y,x)
   Draw the point (y,-x)
   Draw the point (x,-y)
   Draw the point (-x,-y)
   Draw the point (-y,-x)
   Draw the point (-y,x)
   Draw the point (-x,y)
ENDPROC
```

b. CIRCLE-FILL

This command draws four horizontal line segments between some of the eight symmetric points. When a filled circle is to be drawn, CIRCLE-FILL is called by the DRAWCIRCLE command to draw the four horizontal line segments between some of the eight symmetric points for each point on the circle generated by DRAWCIRCLE.

INPUT: X -- X-coordinate of a circle point
       Y -- Y-coordinate of a circle point
OUTPUT: none

```
PROCEDURE   CIRCLE-FILL
   Draw a horizontal line segment from (X,Y) to (-X,Y)
   Draw a horizontal line segment from (Y,X) to (-Y,X)
   Draw a horizontal line segment from (Y,-X) to (-Y,-X)
   Draw a horizontal line segment from (-X,-Y) to (X,-Y)
ENDPROC
```

c. DRAWCIRCLE

This command applies J. Michener's algorithm to calculate the coordinates of the points on the circle and calls CIRCLE-PUT to either draw a circle or a filled circle determined by the value of the fill-flag (FILLFLG).

If the value of the fill-flag is 0, CIRCLE-PUT calls CIRCLE-PTS to plot the eight symmetric points; otherwise, it calls CIRCLE-FILL to draw the four horizontal line segments between some of the eight symmetric points.

INPUT: CX -- X-coordinate of the center point

CY -- Y-coordinate of the center point

R -- Radius of the circle

OUTPUT: none

```
PROCEDURE   DRAWCIRCLE
   Transform the origin to (CX,CY)
   Set x to 0
   Set y to R
   Compute d = 3 - 2 * R
   WHILE  x < y
      Call CIRCLE-PTS to plot eight symmetric points
      IF d < 0 THEN
         Compute  d = d + 4 * x +6
      ELSE
         Compute  d = d + 4 * ( x - y ) + 10
         Decrement y by 1
      ENDIF
      Increment x by 1
   ENDWHILE
   IF x = y THEN
      Call CIRCLE-PTS to plot eight symmetric points
   ENDIF
ENDPROCE
```

Consequently, the CIRCLE and FCIRCLE commands become quite simple. They either set or reset the fill-flag (FILLFLG) and call DRAWCIRCLE to draw the appropriate graph.

Here are more specific details for both CIRCLE and FCIRCLE.

d. CIRCLE

This command draws a circle with center at the given coordinates with radius R.

INPUT: CX -- X-coordinate of the center

CY -- Y-coordinate of the center

R -- Radius of the circle

OUTPUT: none

```
PROCDURE   CIRCLE
   Set the fill-flag to "off"
   Call DRAWCIRCLE to draw the circle
   Position the cursor at the center of the circle
ENDPROC
```

e. FCIRCLE

This command draws a filled circle centered at the given coordinates with the given radius.

INPUT: CX -- X-coordinate of the center

CY -- Y-coordinate of the center

R -- Radius of the circle

OUTPUT: none

```
PROCEDURE   FCIRCLE
   Set the fill-flag to "on"
   Call  DRAWCIRCLE to draw the line segments that fill the
          circle
   Position the cursor at the center of the circle
ENDPROC
```

10. Draw Text String Commands

These commands are used to draw text strings on the display starting at a given point. The text string is either entered by the user interactively at execution-time or put in word definitions for later compilation.

The shape of each character in a string is taken from the standard ATARI character set which is indexed by the Internal Code for the ATARI computer.

a. CHARACTER

This command takes the shape data for the given ATASCII code and draws that character on the display. It is the basis for each text string drawing command.

INPUT: X -- X-coordinate of the given point

Y -- Y-coordinate of the given point

CODE -- ATASCII code for the given character

OUTPUT: none

```
PROCEDURE   CHARACTER
   Convert the ATASCII code into the Internal Code
   Compute the starting address of the shape data for  that
             character
   FOR each byte of shape data
      FOR each bit of that byte
         IF the bit is "on" THEN
            Plot a point at (X,Y)
         ENDIF
         Increment X by 1
      ENDFOR
      Reset X to the original value
      Increment Y by 1
   ENDFOR
ENDPROC
```

b. TEXT

This command draws a text string, entered by user interactively, on the display starting at the given point. Note: the reference position for the first character is the bottom left-hand corner.

INPUT: X -- X-coordinate of the given point

Y -- Y-coordinate of the given point

OUTPUT: none

```
PROCEDURE   TEXT
   Display the prompt message
   Query the use for the string
   FOR each character of that string
      Call CHARACTER to draw that character
      Increment X by 8
   ENDFOR
ENDPROC
```

c. STRING"

This word is an immediate word. At compile time, it compiles (PRINT) and the string from the input stream into the dictionary. At execution-time, it executes (PRINT) to draw the compiled string on the display. Note: the reference position for the first character is the bottom left-hand corner.

INPUT: X -- X-coordinate of the given point
       Y -- Y-coordinate of the given point
OUTPUT: none

```
PROCEDURE   STRING"
   IF the state is compile state THEN
      Compile (PRINT) into the dictionary
      Compile the string from the input stream
   ELSE
      Draw  the compiled string on the display starting  at
                the point (X,Y)
   ENDIF
ENDPROC
```

11. Fill Command

For clarity, some special terms are defined before the fill algorithms are considered. The interior of a region consists of all the pixels within the region that are not on the boundary of the region. A region is called 4-connected if all pixels in the region can be reached, one from the other, by a sequence of the one-pixel moves: up, down, left, or right. The command BFILL in FIGS is concerned with 4-connected regions.

A very simple algorithm to fill a 4-connected region is the Simple Recursive Algorithm(10) which applys recursion to the set of four neighbor pixels after plotting a certain pixel. The primary drawback of this algorithm is that many levels of recursion take place and may cause stack overflow when memory space is limited. A more efficient approach has been developed by A. R. Smith(11).

Smith's algorithm divides the region into runs, which are horizontal groups of adjacent pixels. Only one pixel position is kept for each run to reduce the depth of recursion. The algorithm proceeds as follows: the contiguous horizontal run of pixels containing the starting point is filled in. Then the row just above the filled run is examined from right to left to find the rightmost pixel of that run. These rightmost pixel addresses are stacked. The same is done for the row just below the filled run.

When a run has been processed in this manner, the pixel address at the top of the stack is used as a new starting point. When the stack becomes empty, the algorithm terminates.

The algorithm used in BFILL also uses the idea of runs, but the sequence of both filling a run and finding the seeds for possible runs above and/or below the filled run are totally different from that of Smith's algorithm.

It proceeds as follows: scan each pixel of a run from the starting point towards the left to find the left boundary position of that run. Then fill the run with the current color value. At the same time, check the rows, both above and below, to determine whether some more runs are required; if so, keep the seeds for these runs on the stack. When a run has been processed in this manner, the seed at top of the stack is used as the next starting point. When the stack is empty, the algorithm terminates.

a. BFILL

This command fills a 4-connected region with the current color starting at the given starting point. If the pixel at the starting point already has the same value as the current color number, this command terminates immediately.

```
INPUT:  X -- X-coordinate of the starting point
        Y -- Y-coordinate of the starting point
OUTPUT: none

PROCEDURE  BFILL
   Read the pixel value of the starting point and keep it
             as the old-value
   IF the old-value is different from the fill value THEN
      Initialize the stack
      Push the starting point on the stack
      WHILE the stack is not empty
         Pop the top point as the next starting point
         Call <FILL to find the left boundary and set both
                  uptrig and downtrig to 1
         Call FILL> to plot the pixel values of the run and
                  check for new runs above and below
      ENDWHILE
   ENDIF
ENDPROC
```

b. FILL>

This word is called by the word BFILL. It fills the current run with the current color and checks the rows above and below the current run to determine whether some more runs are required. If more runs are required, FILL> puts the seeds of these runs on the stack.

INPUT: ADDR -- address of the left boundary of the run

PXR -- pixel number of the left boundary point of the current run

X -- X-coordinate of the left boundary of the run

Y -- Y-coordinate of the left boundary of the run

OUTPUT: none

```
PROCEDURE   FILL>
   FOR  each pixel of the run from left to right
      Set the pixel value to the current color number
      IF the value of upper pixel = old-value THEN
         IF uptrig = 1 THEN
            Push the position of upper pixel on the stack
            Set uptrig to 0
         ENDIF
      ELSE
         Set uptrig to 1
      ENDIF
      IF the value of lower pixel = old-value THEN
         IF downtrig = 1 THEN
            Push the position of lower pixel on the stack
            Set downtrig to 0
         ENDIF
      ELSE
         Set downtrig to 1
      ENDIF
   ENDFOR
ENDPROC
```

12. Clear Screen Commands

These commands are used to clear the whole screen or a rectangular region within the screen determined by two opposite-corner points. This is done simply by temporarily changing the current color to the background color and utilizing the FBOX command.

a. BCLEAR

This command clears a rectangular region within the screen determined by two opposite-corner points.

INPUT: X1 -- X-coordinate of the first point
Y1 -- Y-coordinate of the first point
X2 -- X-coordinate of the second point
Y2 -- Y-coordinate of the second point
OUTPUT: none

```
PROCEDURE   BCLEAR
   Temporarily save the current color
   Set the current color to the background color
   Call FBOX to draw a filled rectangular determined by two
             opposite-corner points
   Reset the current color back to the saved color
ENDPROC
```

b. CLEAR

This command clears the whole screen.

INPUT: none

OUTPUT: none

```
PROCEDURE   CLEAR
   Fetch the four boundaries of the terminal window
   Call  BCLEAR to clear a rectangular block whose  extent
            is the terminal window
ENDPROC
```

13. Box Region Moving and Copying Commands

These commands move and copy a rectangular region of screen data to another place on the same screen. The main difference between the two commands in this section is that the move command will fill the source rectangular region with the background color after copying the screen data to another place, but the copy command will not.

a. Horizontal Line Copying

For easier implementation, a rectangular region is divided into horizontal lines. Copying a rectangular region then becomes the process of copying a series of horizontal lines.

For some graphics modes, a screen data byte does not contain exactly one pixel. Furthermore, the starting pixel, for a horizontal line within a source or destination rectangular region, is not always located exactly on a byte boundary. As a result, screen data from one region can not be copied to another region in a byte by byte manner.

An approach for copying a horizontal line proceeds as follows: first, copy the minimum number of bytes of screen data which contain the pixels of the source horizotal line into a working area. Second, shift the pixels, which are in the working area, either to the left or to the right so that they have the same relative alignment within the destination horizontal line. Then put the surrounding pixels of the destination horizontal line into the two end bytes. Finally, copy the bytes of screen data from the working area to the destination horizontal line byte by byte. Figure V-7 illustrates the copying process for a horizontal line from the source to the destination. In FIGS, this work is performed by the command LMOVE.

```
         |----|----|----|----|----|----|----|  Destination line
         |dddd|dddd|dddd|dddd|dddd|dddd|dddd|  before copying
 --------|    |    |    |    |    |    |    |
 |       |____|____|____|____|____|____|____|
 |
 |
 |       |----|----|----|----|----|----|----|
 |       |ssss|sxxx|xxxx|xxxx|xxss|ssss|ssss|  Source line
 |       |    |    |    |    |    |    |    |
 |       |____|____|____|____|____|____|____|
 |            |                   |
 |            |  copy to working  |
 |       |----|      area    |----|
 |       V                   V
 |       |----|----|----|----|----|----|----|  Working area
 |       |sxxx|xxxx|xxxx|xxss|    |    |    |  before
 |       |____|____|____|____|____|____|____|  shifting
 |         |                  |
 |         |-| shift pixels   |--|
 |           V                   V
 |       |----|----|----|----|----|----|----|  Working area
 |       |  sx|xxxx|xxxx|xxxx|ss  |    |    |  after
 |       |____|____|____|____|____|____|____|  shifting
 |
 |         handle surrounding
 |              pixels
 |       |----|----|----|----|----|----|----|  Working area with
 |-->    |dddx|xxxx|xxxx|xxxx|dddd|    |    |  surrounding pixel
         |____|____|____|____|____|____|____|
         |         copy  to           |
         |---------| destination      |---------|
                   V                            V
         |----|----|----|----|----|----|----|
         |dddd|dddd|dddx|xxxx|xxxx|xxxx|dddd|  Destination line
         |____|____|____|____|____|____|____|  after copying
```

Figure V-7 Copying a Horizontal Line

```
INPUT:  none
OUTPUT: none

PROCEDURE    LMOVE
   Copy  screen data bytes  from the source horizontal line
            to the working area
   IF the first pixel of the destination horizontal line is
            in  a higher-order pixel position than is  the
            first pixel of the source horizontal line THEN
      Shift  the pixels in the working area to the right to
               match  the  alignment  of  the  destination
               horizontal line
   ELSE
      Shift  the pixels in the working area to the left  to
               match  the  alignment  of  the  destination
               horizontal line
   ENDIF
   Put the surrounding pixels of the destination horizontal
               line in the two end bytes
   Copy the bytes from the screen data working area to  the
               destination hosizontal line
ENDPROC
```

Since the basic components for copying rectangular regions have been presented, the two primary commands for copying and moving rectangular regions can be discussed.

b. RCOPY

This command copies the contents of a rectangular region to another place within the same screen. The contents of the source region remain unchanged after the execution of this command.

```
INPUT:   SX1 -- X-coordinate  of  the first  point  of  the
                source retangular region
         SY1 -- Y-coordinate  of the  first  point  of  the
                source retangular region
         SX2 -- X-coordinate of the  second  point  of  the
                source retangular region
         SY2 -- Y-coordinate  of  the second point  of  the
                source retangular region
         DX  -- X-coordinate  of the starting point for the
                destination rectangular region
         DY  -- Y-coordinate  of the starting point for the
                destination rectangular region
OUTPUT:  none

PROCEDURE  RCOPY
   Prepare for the LMOVE command
   FOR  each horizontal line within the source  rectangular
                  region
      Call LMOVE to copy that horizontal line
   ENDFOR
ENDPROC
```

c. RMOVE

This command copies the contents of a rectangular region to another place within the same screen and fills the source rectangular region with the background color.

INPUT: SX1 -- X-coordinate of the first point of the source retangular region

SY1 -- Y-coordinate of the first point of the source retangular region

SX2 -- X-coordinate of the second point of the source retangular region

SY2 -- Y-coordinate of the second point of the source retangular region

DX -- X-coordinate of the starting point for the destination rectangular region

DY -- Y-coordinate of the starting point for the destination rectangular region

OUTPUT: none

```
PROCEDURE   RMOVE
   Prepare for the LMOVE command
   FOR  each horizontal line within the source  rectangular
                    region
      Call LMOVE to copy that horizontal line
      Fill  the source horizontal line with the  background
                    color
   ENDFOR
ENDPROC
```

14. Scale or Zoom Commands

These commands provide the capability of magnifying a certain part of the screen by a factor of 2 or 4.

Normally, zooming is done by mapping the zoom window, which defines the extent of a certain picture to be seen, to a certain viewport, which defines the area of the display where the zoomed picture is to be shown. To do this mapping, the following general mapping equations are normally used.

$$Xv = Xv.min + \frac{Xv.max - Xv.min}{Xw.max - Xw.min} (Xw - Xw.min)$$

$$Yv = Yv.min + \frac{Yv.max - Yv.min}{Yw.max - Yv.min} (Yw - Yw.min)$$

These equations are quite powerful; they can be applied to any size of zoom window and any size of viewport as long as the arithmetic computations are valid. However, they involve floating point arithmetic.

The ANTIC graphics modes 8, A and E have the same numbers of colors (four), the resolution of mode A is two times that of mode 8, and the resolution of mode E is two times that of mode A. This means that the pixel size of mode A is two times that of mode E and the pixel size of mode 8 is four times that of mode E.

Consequently, zooming can be done by directing ANTIC to fetch screen data for display lists of mode 8 or mode A from the screen data RAM of mode E. Figure V-8 shows the extent of zoom windows for modes 8, A and E.

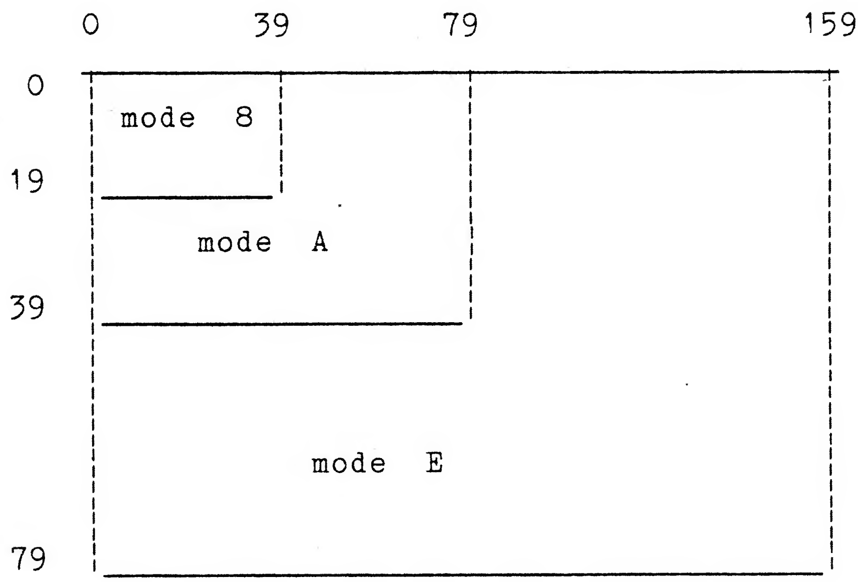


Figure V-8 Extent of Zoom Windows

Since the required screen data for each mode line of either mode 8 or mode A is less than that of mode E, the screen data required by the zoom window becomes not contiguous. A special display list must be prepared to direct the ANTIC to fetch these noncontiguous screen data for each mode line of either mode 8 or mode A. Each mode instruction of this special display list is created with a load memory scan option. In FIGS, the word !ZOOM-DLS is used to prepare that kind of display list.

Furthermore, it is easy to move the zoom window around by simply directing ANTIC to fetch screen data from different parts of screen data RAM; that is, to change the load memory address for each mode line in the display list.

Because these three modes are all 4-color modes and four pixel values are maintained in one byte, it seems impossible to have a zoom window starting from a point with a horizontal coordinate that can not be divided by 4. Fortunately, this problem can be solved by using the fine scrolling technique. FLUSHDL moves the zoom window by changing the load memory addresses for each mode line in the display list and using fine scrolling.

a. !ZOOM-DLS

For the zoom mode, this command creates a special display list which contains a load memory scan option for each mode instruction.

b. FLUSHDL

This commmand moves the zoom window to a place pointed to by the given point. This is done by changing the load memory address for each mode line in the display list and applying fine scrolling.

INPUT: X -- X-coordinate of the upper left-hand corner of the zoom window

OUTPUT: Y -- Y-coordinate of the upper left-hand corner of the zoom window

```
PROCEDURE   FLUSHDL
   Transform the X-Y coordinate to a screen data memory
         location and a pixel number
   Compute a proper fine scrolling value using the pixel
         number
   FOR each load memory scan address in the display list
      Put the screen data memory location in that load
            memory scan address
      Increment the screen data memory location by bytes-
            per-line in this mode.
   ENDFOR
ENDPROC
```

c. >ZOOMXY

This command tries to move the zoom window so that the cursor position can be at the center of that window; if this is not possible, the zoom window will touch the boundary of the logical window and will have the cursor somewhere within the zoom window. It transforms the given X-Y coordinates to zoom X-Y coordinates after moving the zoom window.

15. Player Cursor Manipulation Commands

These commands provide the capabilities to use ATARI players. Please refer to chapter II for more information about player-missile graphics.

a. SINGLE-SIZE, DOUBLE-SIZE, QUAD-SIZE

These three commands set the player size to either single-size (8 color clock width), double-size (16 color clock width) or quadruple-size (32 color clock width) for the given player.

b. SINGLE-LINE, DOUBLE-LINE

These two commands set the player line size option to either single-line or double-line.

c. PLAYER-ON, PLAYER-OFF

PLAYER-ON directs the CTIA to start fetching the player shape data from the player-missile base area. PLAYER-OFF directs the CTIA to stop fetching all player shape data.

d. PLAYER-CLEAR

This command clears the player shape data for the current player.

e. SETPFIG

This command assigns a defined player shape and its reference position to a given player.

f. SETPLAYERS

This command reserves a block of memory for the player-missile base area and sets the default values for each player.

g. USEPLAYER

This command selects the given player number as the current player number and sets the necessary data for that player.

h. PLAYER

This command puts the player shape data in the player shape data area for the current player and sets the horizontal position register to the proper value. When the player display is turned on by PLAYER-ON, it displays the player shape at the given position on the screen.

INPUT: X -- X-coordinate of the given player position

Y -- Y-coordinate of the given player position

OUTPUT: none

```
PROCEDURE   PLAYER
   Transform the X-Y coordinates to the coordinates used
             by the player
   Store the X-coordinates used by the player in the
             horizontal position register
   Move the player shape data to the player shape data area
             for the current player
ENDPROC
```

16. Disk File Management Commands

These commands provide a simple disk file management environment that supports the following functions:

- Display the directory of a disk
- Save the screen data as a file on a disk
- Save the display list in a disk file
- Reload the screen data and/or display list from a disk file
- Delete a disk file
- Format a new disk

The basic component of the file management system is a block (a sector on the ATARI) which is 128 bytes in length. The disk is divided into three categories:

- Master directory sector
- File directory sectors
- Data sectors

Once a new disk has been formated by the FORMAT command, the data sectors are linked together into a free sector list and the directory sectors are linked together to form the directory. The master directory sector keeps the pointers and counters for the free sector list and the head of the directory sectors. It also keeps track of the total number of sectors in this disk and the number of sectors currently being used. Figure V-9 illustrates the master directory sector layout. Figure V-10 illustrates the data sector layout. Figure V-11 illustrates the file directory layout. Figure V-12 illustrates file directory entry layout. Figure V-13 illustrates the relationship among the three kinds of sectors.

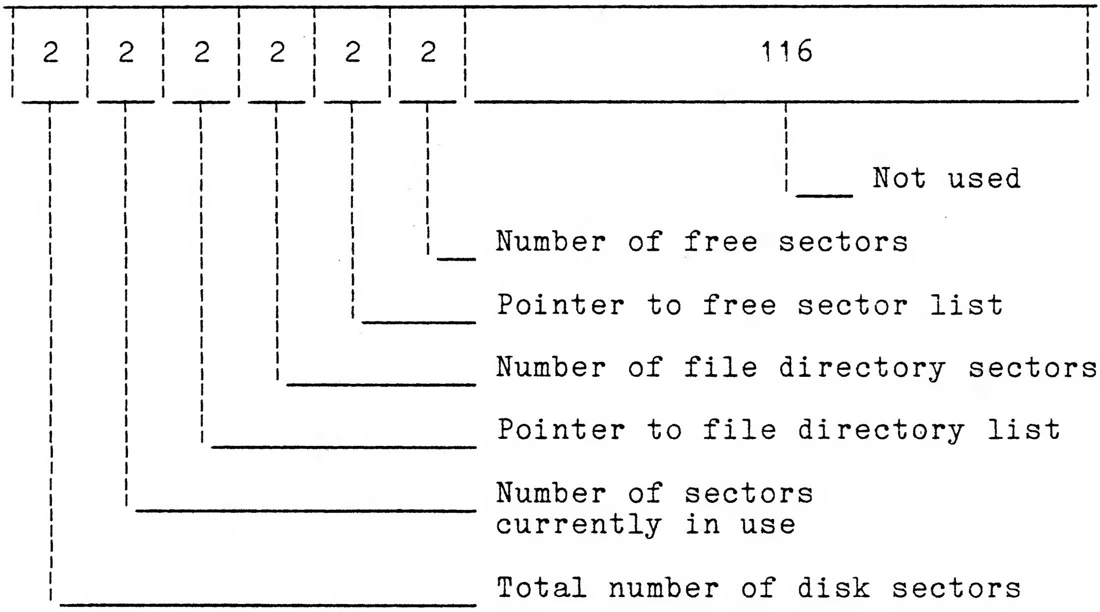


Figure V-9 Master Directory Sector Layout

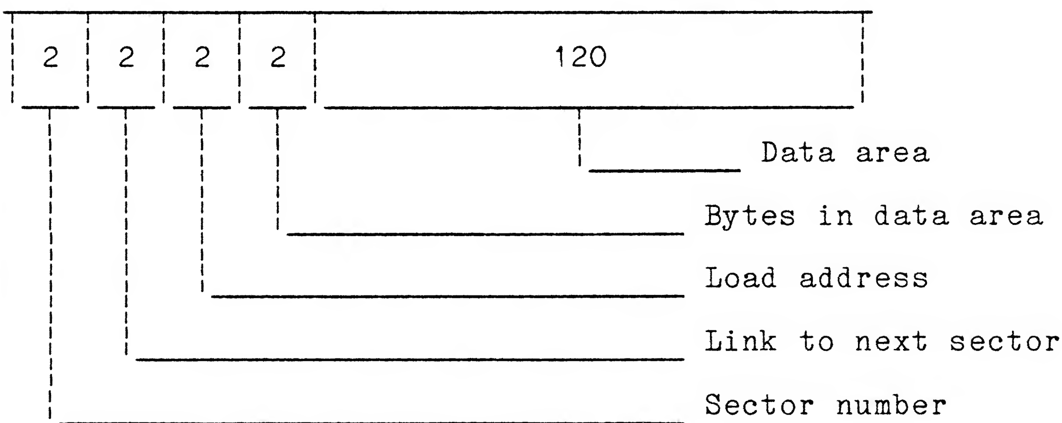


Figure V-10 Data Sector Layout

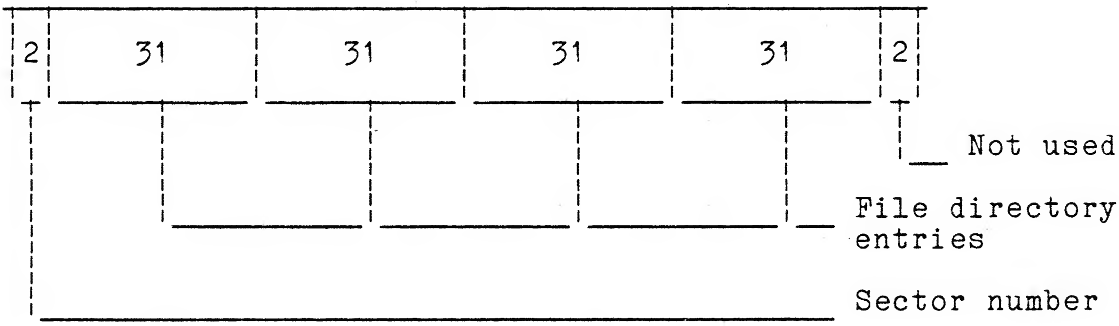


Figure V-11 File Directory Layout

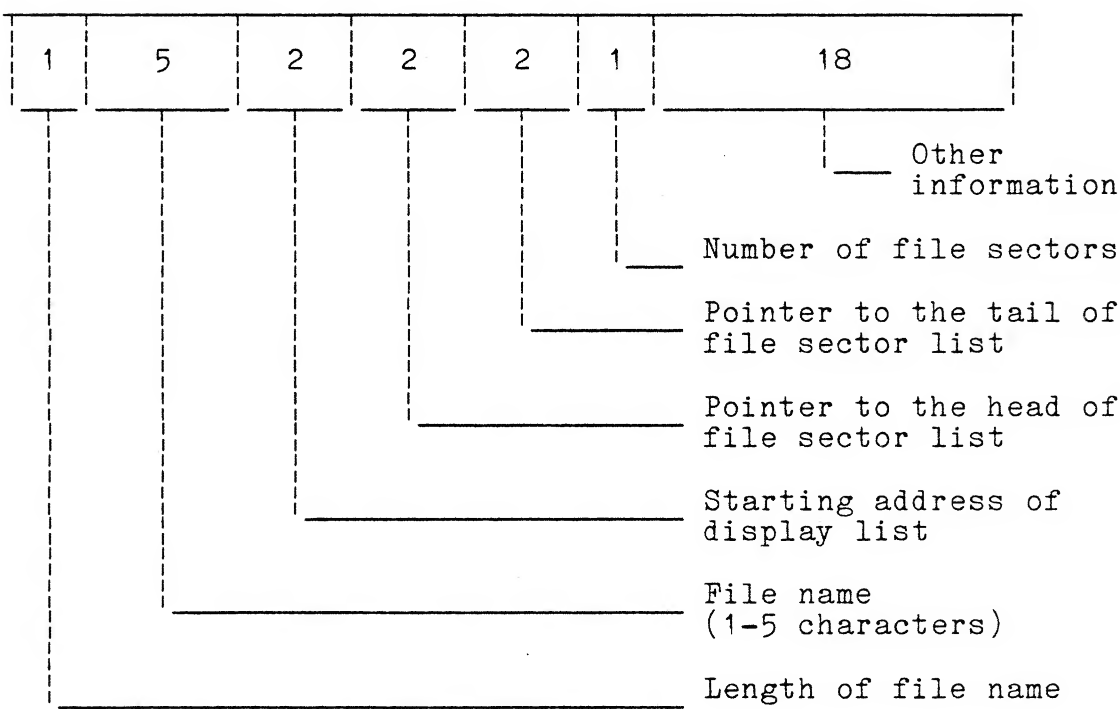


Figure V-12 File Directory Entry Layout

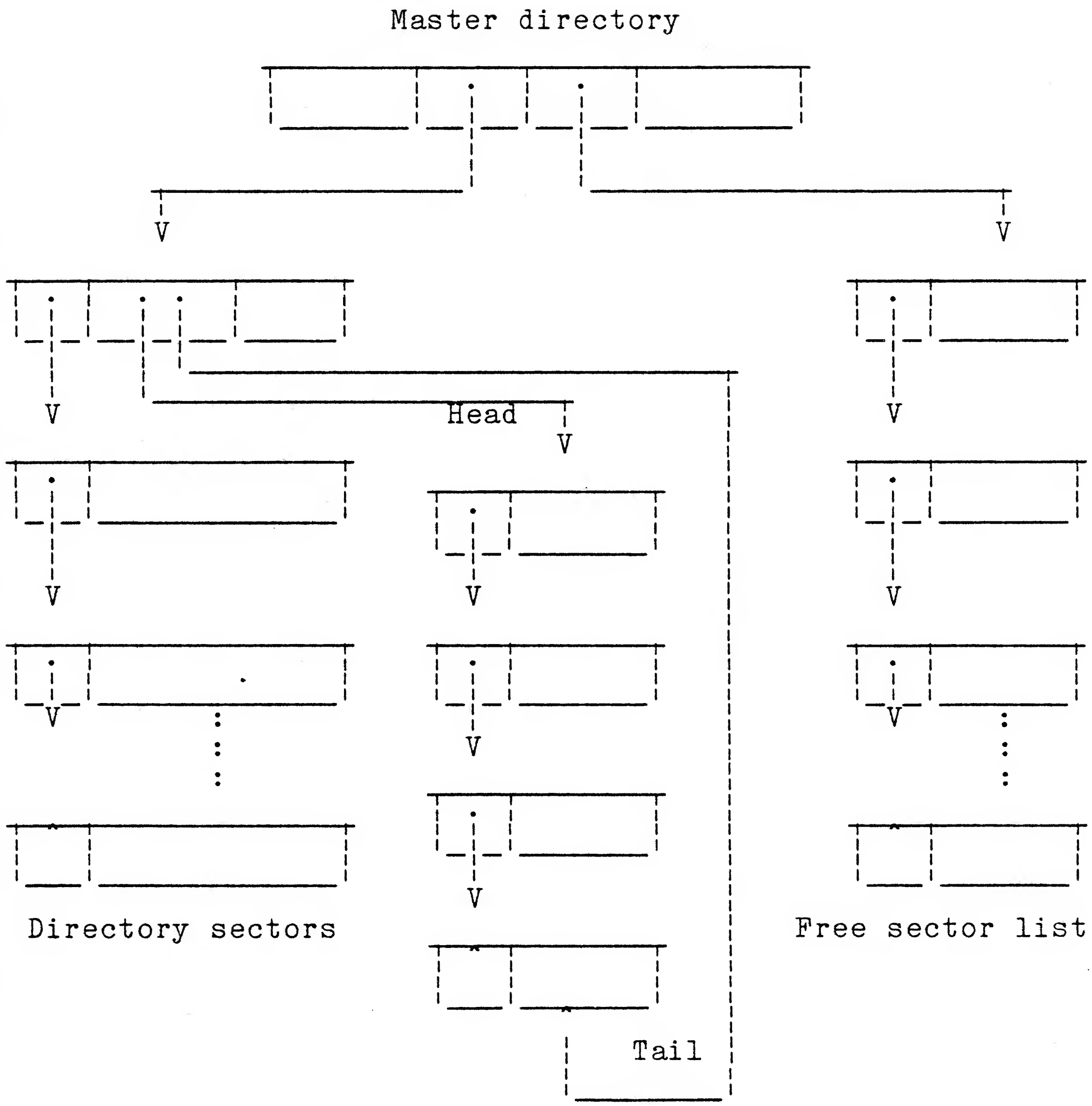


Figure V-13 Relationship Among the Three Kinds of Sectors

a. DIR

This command lists the filenames of the existing files. It provides the capability to look at the filenames for the existing files.

INPUT: none

OUTPUT: none

```
PROCEDURE  DIR
   FOR each file directory sector
      Read this sector
      FOR each file directory entry in this sector
         IF the length of filename > 0 THEN
            Display the file name
         ENDIF
      ENDFOR
   ENDFOR
ENDPROC
```

b. STORE-DATA

Depending on the value of DDFLAG and DLFLAG, this word is called by (GSAVE) to save a display list and/or screen data as a disk file. It is the basis for (GSAVE).

INPUT: none

OUTPUT: none

```
PROCEDURE   STORE-DATA
   Set the display list pointer to the starting location of
             the current display list
   WHILE  the display list pointer does not equal the  last
             byte of the display list
      Get  one  display list instruction from  the  display
                list
      IF the instruction is a blank line instruction THEN
         Increment the display list pointer
      ELSE
         IF the instruction is either a load memory scan or
                   a jump vertical blank instruction THEN
            Store  the previous unsaved part of the  screen
                      data
            Increment the display list pointer by 3
            IF  the  instruction is a jump  vertical  blank
                      instruction THEN
               Store  the  previous  unsaved  part  of  the
                         display list
               Indicate the end of the display list
            ENDIF
         ELSE
            Increment the display list pointer by 1
```

```
        ENDIF
      ENDIF
      IF the length of the previous unsaved screen data  is
                 more than 119 bytes THEN
         Store 120 bytes of screen data on disk
      ENDIF
      IF the length of the previous unsaved display list is
                 more than 119 bytes THEN
         Store 120 bytes of screen data on disk
      ENDIF
   ENDWHILE
ENDPROC
```

c. GSAVE

Depending on the values of DDFLAG and DLFLAG, this command checks whether a specific filename is duplicated; if not, it calls (GSAVE) to save the display list and/or screen data as a disk file.

d. SAVE-DL

If the filename is unique, this command saves the display list in a file with this name. It simply resets the DDFLAG to 0, sets the DLFLAG to 1 and calls GSAVE to complete the work.

e. SAVE-DD

If the filename is unique, this command saves the screen data in a specific file. It simply sets the DDFLAG to 1, resets the DLFLAG to 0 and calls GSAVE to complete the work.

f. SAVE-ALL

If the filename is unique, this command saves both the screen data and display list in the specific file. It simply sets boths DDFLAG and DLFLAG to 1 and calls GSAVE to complete the work.

g. GLOAD

This command reloads the screen data and/or display list from the given file. Since every data sector of a file keeps not only the data itself but also its load address, the load process is quite simple; read each sector of the file and move the data into memory starting from the load address.

h. DELETE

This command deletes the given filename from the file directory and reassigns the used data sectors to the free sector list.

i. FORMAT

This command formats a disk with the format necessary for this simple disk file management system. Actually, it links the file directory sectors into the file directory sector list, it links the data sectors into the free sector list, and it writes the necessary initial values in the master directory sector.

17. KoalaPad Interfacing Commands

These commands read the values of the stylus position, along with the left-hand right-hand buttons of the KoalaPad Touch Tablet. Please refer to chapter III for more information about the KoalaPad Touch Tablet.

a. PADXY

This command reads the X-Y coordinates of the stylus and puts them on top of the parameter stack.

b. LBTN

This command reads a value from the left button of the KoalaPad Touch Tablet and puts it on top of the parameter stack.

c. RBTN

This command reads a value from the right button of the KoalaPad Touch Tablet and puts it on top of the parameter stack.

d. >SCRXY

This command transforms the values read from the stylus position of the KoalaPad Touch Tablet to the normal X-Y coordinates for the current graphics mode.

18. Display List Interrupt Handling Commands

These commands aid the user to handle display list interrupts.

A display list interrupt is a non-maskable interrupt. The process begins when the ANTIC encounters a display list instruction with its interrupt bit (bit 7) set. ANTIC waits until the last scan line of the current mode line is displayed. It then refers to the non-maskable interrupt enable register to see whether the display interrupt bit has been set. If that bit is not set, ANTIC goes back to its normal work; otherwise, an interrupt to the 6502 occurs. The 6502 then vectors through the non-maskable interrupt vector (NMI) to an interrupt service routine in the OS. This service routine first determines the cause of the interrupt. If the interrupt is indeed a display list interrupt, the routine vectors through addresses 200 and 201 (hex) to a display interrupt service routine.

In the FORTH environment, the service routine can be written as a low-level code word with the code field address (CFA) of that word as its entry point. This routine must end with a RTI (return from interrupt) instruction.

Once the service routine has been prepared, the address of its entry point can be put in addresses 200, and 201 (hex). Finally, the user can enable the display list interrupt routine by setting the enable bit for the display list interrupt.

a. DLI-ON

This command enables the display list interrupt routine by setting the display list interrupt enable bit in the NMIEN register to 1.

b. DLI-OFF

This command disables the display list interrupt routine by resetting the display list interrupt enable bit in NMIEN register to 0.

c. SETDLIV

This command put the address of the entry point for the display list interrupt service routine, which is on top of the parameter stack, in the display list interrupt vector at 200, 201 (hex).

VI. A FORTH INTERACTIVE GTAPHICS EDITOR

The purpose of the Forth Interactive Graphics Editor (FIGE) is to provide capabilities for creating graphics on the display surface of a television by using the KoalaPad Touch Tablet and the ATARI keyboard. The human factors considered during the design of FIGE were

- Make it simple for the user to use
- Give the user more than one chance

The first factor requires a simple man-machine interface that is not only easy to use but is also easy to learn. In FIGE, a simple menu driven interface is achieved with the KoalaPad and the ATARI keyboard. It is very simple both to use and to learn.

The second factor allows the user to gracefully recover from mistakes. The FIGE performs boundary checking for all graphics draw commands. It also requires user acknowledgement for such critical destructive commands as clearing a whole screen, formatting a disk, and exiting from the system without saving screen data.

For the sake of convenience, the main menu of FIGE, which first appeared in chapter I, is presented again in Figure VI-1.

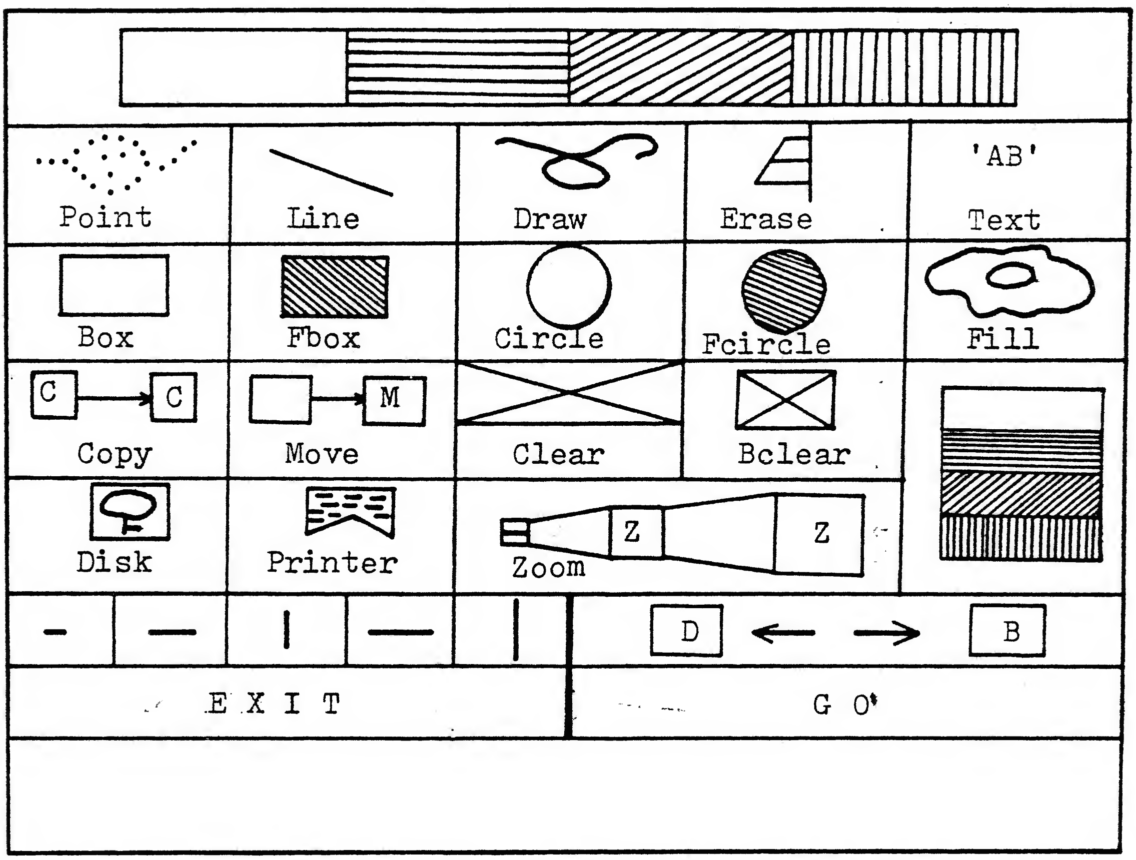


Figure VI-1 Main Menu of FIGE

A. System Overview

The FIGE has been developed as an application oriented extension of FIGS. It utilizes the primitive commands provided by FIGS to perform most of its functions. Since FIGE is a menu driven graphics editor, all graphics features are selected and performed through a menu. As a result, an efficient pointing device was required for selecting entries from the menus. The KoalaPad Touch Tablet was chosen as the primary pointing device because of its significant features that aid menu selection. Refer to chapter III for more information about this device.

Before descriptions of the system structure are presented, a summary of the features provided by the FIGE are listed below:

- Plot a point at the present cursor position
- Draw a line between two points
- Draw points determined by the trace of the cursor
- Erase points
- Draw a text string
- Draw a rectangle
- Draw a filled rectangle
- Draw a circle
- Draw a filled circle
- Fill a connected closed region with the current color

- Copy a rectangular region to another place
- Move a rectangular region to another place
- Clear the whole screen
- Clear a rectangular region
- Display a color menu and select colors from it
- Display a disk management menu and provide functions such as displaying the directory, saving screen data, loading screen data, deleting a file, and formatting a disk
- Zoom the picture to either normal, double or quadruple size
- Select brushes to draw or erase with
- Adjust the luminance of the current color
- Select a color number from the available (four) color numbers

A user's guide for FIGE is presented in the next section.

Since the FIGS is a menu driven system, system control is based on the user selecting entries from a menu. The processing sequence listed below is executed repeatedly by FIGS.

Show the menu

Get a command

Execute that command

The main structure chart for FIGE is shown in Figure VI-2 while the subordinate structure charts are shown in Figure VI-3, Figure VI-4, and Figure VI-5.

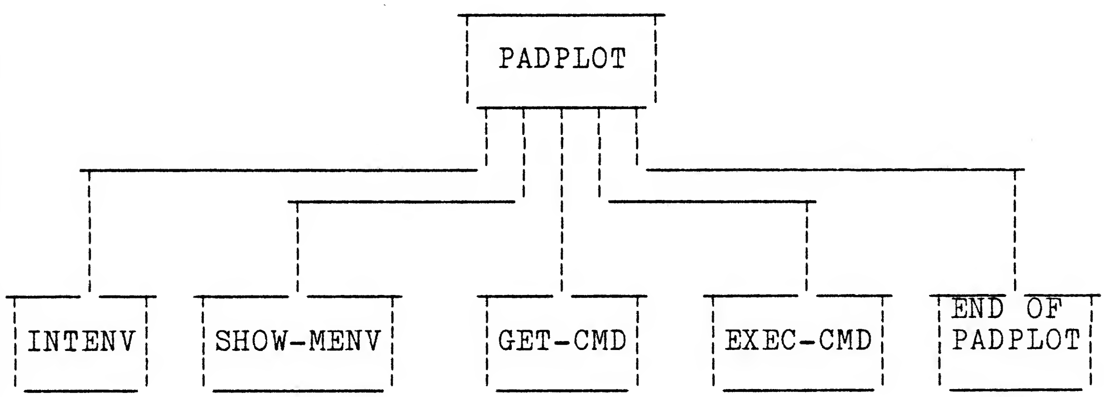


Figure VI-2 Main Structure Chart for PADPLOT

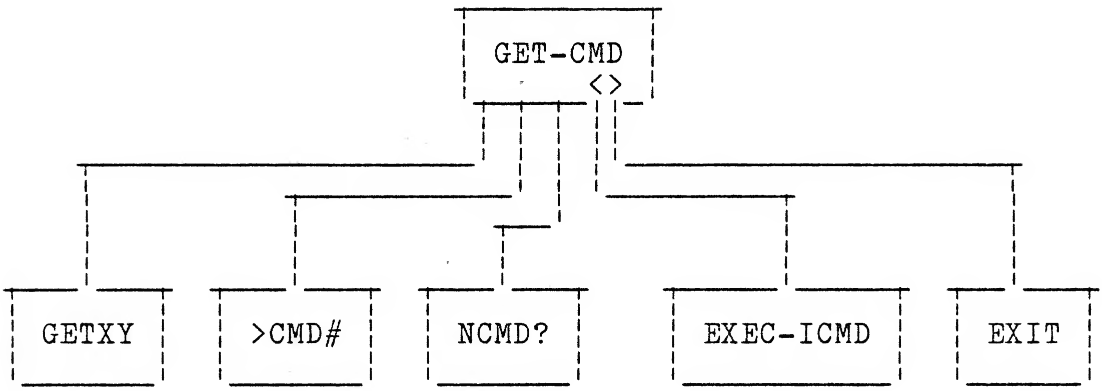


Figure VI-3 Structure Chart for GET-CMD

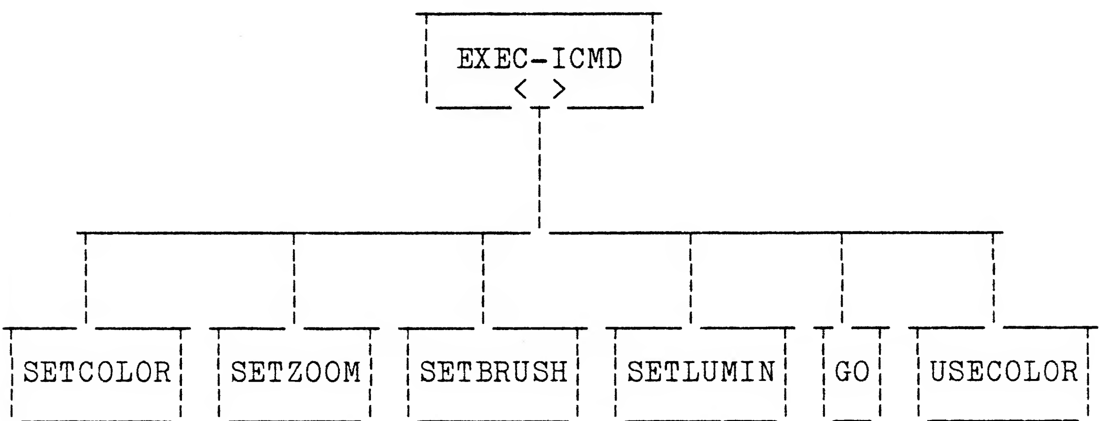


Figure VI-4 Structure Chart for EXEC-ICMD

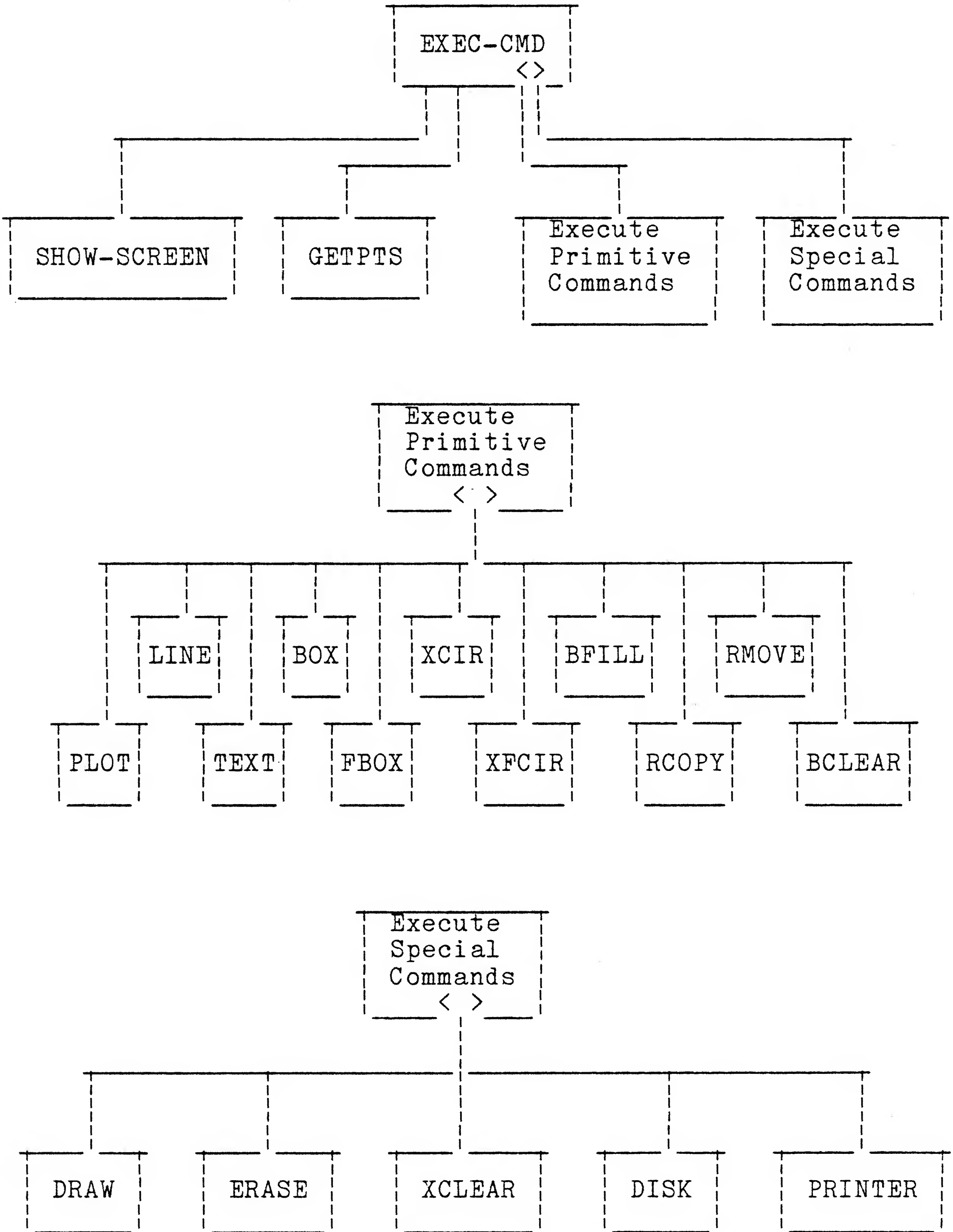


Figure VI-5 Structure Charts for EXEC-CMD

B. Detail Description of Commands

In this section, the important words from FIGE are described in detail. They are presented in the top-down sequence as they appear in the system structure charts presented in the last section.

1. PADPLOT

This command is the only command that can be used to enter FIGE when the CONTEXT vocabulary is DEMO. It is at the top level of FIGE. See the structure chart shown in figure VI-2. PADPLOT calls INITENV first to display the title page of FIGE, to prompt the user for entering "GO", to load the main menu into RAM from the disk, and to initialize the required environment. It then repeatedly executes the sequence: show the menu, get a command and execute that command, until the user chooses to leave the system by selecting the command EXIT. Finally, it informs the the user that PADPLOT has been exited and returns to the FORTH environment.

INPUT: none

OUTPUT: none

```
PROCEDURE   PADPLOT
   Call INITENV to load the main menu and to initialize the
            required environment
   Set exitflag to 0
   IF the initialization process was successful THEN
      UNTIL exitflag = 1
         Call SHOW-MENU to display the main menu
         Call GET-CMD to allow user to select commands with
                  the  KoalaPad Touch Tablet  stylus  from
                  the main menu
         IF  the attempt at selecting a  command  succeeded
                              THEN
            Call EXEC-CMD to execute that command
         ENDIF
      ENDUNTIL
   ENDIF
ENDPROC
```

2. INITENV

This word is called by PADPLOT. It displays the title page for FIGE, prompts the user to enter "GO" from a keyboard to load the main menu, and initializes the environment. The content of the title page is shown in Figue VI-6.

```
              W E L C O M E   T O
                  P A D P L O T
                       BY
             J E R - M I N G   L E E
                     1 9 8 5
 ----------------------------------------
 I N S E R T   S C R E E N   D I S K
 T Y P E  ' G O '  T O   S T A R T
```

Figure VI-6 Title Page of FIGE

INPUT: none

OUTPUT: Result flag

1: "GO" was typed and the main menu was successfully loaded

0: only the <RETURN> key was pressed. The user wanted to leave the system

```
PROCEDURE   INITENV
   Set the color registers to their default values
   UNTIL  "GO" is typed or only the <RETURN> key is pressed
      Display the title page
      Query  the user to either enter "GO" or to press  the
                <RETURN> key
      IF "GO" was typed THEN
         Inform the user to wait
         Call GLOAD to load the main menu from a disk  file
                   named "GO"
         Reset the current zoom, the current brush, and the
                   current command to their default values
         Set the result flag to 1
      ELSE
         IF only the <RETURN> key was pressed THEN
            Set the result flag to 0
         ENDIF
      ENDIF
   ENDUNTIL
ENDPROC
```

3. SHOW-MENU

This word displays the main menu on the screen, marks the current color and the current brush with players 1 and 2, and displays the names of the current command and the current zoom. It is called by PADPLOT. Figure VI-1 illustrates the main menu.

4. GET-CMD

This word allows the user to select one of the commands from the main menu by pressing the stylus on the KoalaPad Touch Tablet and pushing the appropriate button. It can also execute immediate commands, which do not directly operate on the main graphics screen. Such commands include changing the colors, selecting the current color, selecting the current brush, selecting the current zoom mode, changing the luminance of the current color, directing FIGE to exit and directing FIGE to execute the currently selected command.

INPUT: none

OUTPUT: CMD# -- command number

1 -- flag indicating selected command is OK

or 0 -- flag indicating that the EXIT immediate command has been executed and that the execution of the current command is not necessary

```
PROCEDURE   GET-CMD
   UNTIL   either  the  immediate  command EXIT  or  GO  is
                executed
      UNTIL   a  valid  command is selected  and  the  left
                   button  of  the KoalaPad Touch Tablet  is
                   pressed
         Get a position from the Koalapad Touch Tablet
         Call  >CMD#  to convert the position to  a command
                     number
         IF the command number is valid THEN
            Display the name of that command as a candidate
                       command
         ENDIF
      ENDUNTIL
      IF the command is an immediate command THEN
         Execute that command
      ELSE
         Set the current command to that selected command
         Display the name  of that command as the  current
                    command
      ENDIF
   ENDUNTIL
ENDPROC
```

5. EXEC-CMD

This word is called by PADPLOT. It executes the command selected by GET-CMD. This is done by means of vectored execution. The vector for each command is kept in a table called the command table. The vector for a command contains the code field address (cfa) of that word. When a command is selected for execution, the cfa for this command is executed by the FORTH word EXECUTE.

Table VI-1 illustrates the contents of the command table. This table consists of several arrays, namely CMD-CFA, CMD-ATTR, and CMD-NAME. CMD-CFA contains the cfas of the commands. COM-ATTR contains the attributes of the commands. CMD-NAME contains the names of the commands.

Command number	Command name	Word in FIGE	Command attribute (hex)
0	POINT	PLOT	1
1	LINE	LINE	2
2	DRAW	DRAW	0
3	ERASE	ERASE	0
4	TEXT	XTEXT	1
5	BOX	BOX	2
6	FBOX	FBOX	2
7	CIRCLE	XCIR	2
8	FCIRCLE	XFCIR	2
9	FILL	BFILL	1
10	RCOPY	RCOPY	3
11	RMOVE	RMOVE	3
12	CLEAR	XCLEAR	0
13	BCLEAR	BCLEAR	2
14	SETCOLOR	SETCOLOR	10
15	DISK	DISK	0
16	PRINTER	PRINTER	0
17	SETZOOM	SETZOOM	10
18	SETZOOM	SETZOOM	10
19	SETCOLOR	SETCOLOR	10
20	SETBRUSH	SETBRUSH	10
21	LUMINANCE	SETLUMIN	10
22	EXIT	EXIT	10
23	GO		10
24	USECOLOR	UCOLOR	10

Table VI-1 FIGE Command Table

```
INPUT:  CMD# -- command number selected by GET-CMD
           1 -- flag indicating the selected command is OK
   or      0 -- flag indicating that the execution of  the
                current command is not necessary
OUTPUT: none


PROCEDURE   EXEC-CMD
   Get  the cfa  of  the current  command  from  the  array
              CMD-CFA
   IF the current command is a primitive command THEN
      Call SHOW-SCREEN to display the main graphics screen
      UNTIL  the right button of the KoalaPad Touch  Tablet
                 is pressed
         Vector  to either ONE-POINT,  TWO-POINTS or THREE-
                    POINTS  to  get  the correct  number  of
                    points needed by the current command
         IF the points were obtained THEN
            Execute the primitive command
         ENDIF
         Clear the base area of the players
      ENDUNTIL
   ELSE
      Execute  the current special command
   ENDIF
ENDPROC
```

6. SETCOLOR

This command displays a color menu which overlays the left-hand side of the main menu and allows the user to select a color for the current color. The color menu is a collection of colored bands. It is shown by means of player 3 in quadruple size. The purpose of using a player as a color menu is to make it easier to overlay the color menu on the main menu. A display list interrupt is used to show 16 different colors on the same screen. First, the 16 different color values are prepared in an array called COLRPOOL. Second, a display list is prepared with the interrupt bit set on every other ten mode lines. Third, a display list interrupt service routine is prepared that fetches a color value from COLRPOOL and stores it in the color register of player 3 whenever a display interrupt occurs. Finally, the interrupt service routine is enabled.

In FIGE, all four steps are done before SETCOLOR is called. Consequently, SETCOLOR shows the color menu by simply moving the player to the proper place from the previous hidden place. When selecting a color from the color menu, the user can also change the luminance of the current color.

INPUT: none

OUTPUT: none

```
PROCEDURE   SETCOLOR
   Show the color menu by moving player 3 to position (0,0)
   UNTIL  the  selected position is not for the color  menu
             and not for changing the current color and not
             for changing the luminance
      Select a position on the menu screen
      IF  the selected position is for the color menu THEN
         Store  the  selected  color in the  current  color
                 register
      ELSE
         IF  the  selected position is  for  adjusting  the
                 luminance THEN
            Adjust the luminance
         ELSE
            Set  the  current color to the  selected  color
         ENDIF
      ENDIF
   ENDUNTIL
   Move player 3 to a hidden place
ENDPROC
```

7. DISK

This command displays a disk operation menu that allows the user to select one of the disk operations with the KoalaPad Touch Tablet. Figure VI-7 illustrates the disk operation menu.

```
DISK OPERATION MENU
  >  DIRECTORY
  >  SAVE SCREEN
  >  LOAD SCREEN
  >  DELETE FILE
  >  FORMAT DISK
  >  RETURN
```

Figure VI-7 Disk Operation Menu

INPUT: none

OUTPUT: none

```
PROCEDURE   DISK
   UNTIL the 'RETURN' option is selected or the right
            button of the KoalaPad Touch Tablet is pressed
      Display the disk operation menu
      UNTIL a valid option is selected
         Call ONE-POINT to get one point from the screen
         IF this point is a valid point THEN
            Convert the coordinates of that point to a menu
                      option
            IF this is a valid option THEN
               Vector  to the command corresponding to this
                         option
               Execute that command
            ENDIF
         ENDIF
      ENDUNTIL
   ENDUNTIL
ENDPROC
```

8. DRAW

This command draw points determined by the trace of the cursor. The cursor can correspond to one of five brushes that have different sizes and shapes. Different brushes are selected by the SETBRUSH command. When the cursor is moved to a new point, this command draws lines between the respective points corresponding to the current brush. In order to draw points with different types of brushes, the DRAWLINE command is vectored to the appropriate draw command.

Because noise is sometimes detected as points by the KoalaPad Touch Tablet, the word GOODPOINT? is used to check for noise. This is done by continually comparing consecutive pairs of points that are read by the KoalaPad; if the second point is more than five points away from the first point, then the second point is treated as noise, that is, it is ignored.

INPUT: none

OUTPUT: none

```
PROCEDURE   DRAW
   WHILE  the right button of the KoalaPad Touch Tablet  is
             not pressed
      Call SETPOINT to set the first point
      WHILE the left button of the KoalaPad Touch Tablet is
                pressed
         Call SETPOINT to set the second point
         Call GOODPOINT? to check for noise
         IF the second point is a good point THEN
            Call DRAWLN to draw a line from the first point
                      to  the  second  point  and  set  the
                      current point to be the second point
         ENDIF
      ENDWHILE
   ENDWHILE
ENDPROC
```

C. User's Guide

In this section, a complete description for each selection in FIGE is presented. For convenience, the appropriate figure from the main menu is presented with each selection.

1. Getting Started

- Boot up the system from the FIGE boot diskette which contains fig-FORTH, FIGS, and FIGE.
- Type "GRAPHICS" to switch to the GRAPHICS vocabulary. This vocabulary contains FIGS.
- Type "DEMO" to switch to the DEMO vocabulary. This vocabulary contains FIGE.
- Type "PADPLOT" to start FIGE. The title page for FIGE will then appear on the display screen. Figure VI-6 shows the contents of this title page.
- Insert into the drive a FIGE file system diskette which contains the file to be operated on.
- Type "GO" to load the main menu. After a few seconds, the main menu illustrated in Figure VI-1 will be displayed.

2. Using the Main Menu

The main menu for FIGE consists of four sections, namely, the color section, the command section, the message section, and the color menu section.

- Color section (top): This section contains the current colors that are in the four color registers.
- Command section (middle): This section contains entries for selecting graphics commands, a set color command, a set brush command, and a set zoom mode command.
- Message section (bottom): This section dynamically displays messages to the user, such as the name of the current command, the name of the candidate command, and the name of the current zoom mode.
- Color menu section: This section is shown only when the SETCOLOR command is activated. The left-hand side of the command section is overlayed when the color menu is being shown.

To make a selection from the menu, follow the steps listed below:

- Press the stylus on the touch tablet. A cursor will appear on the display screen.
- Move the stylus so that the position of the cursor is on the entry to be selected. When this occurs, the name of the candidate command will be displayed in the message section with the label "NEW:".
- Press the left-hand button when the entry is correctly positioned. When this entry has been selected, the corresponding command name will be displayed in the message section with a label "CMD:". Note: not all entries will effect the current command status.

3. Taking a Glance at the Graphics Screen

The graphics screen consists of two main sections, namely, the graphics section and the message section.

- Graphics section: This section is based on ANTIC mode D which has 4 colors and 160 by 80 resolution.
- Message section: This section dynamically displays messages to the user, such as the current command (labled as "CMD:") and the current cursor coordinate (labled as "X,Y:").

Figure VI-8 illustrates the layout of the graphics screen.

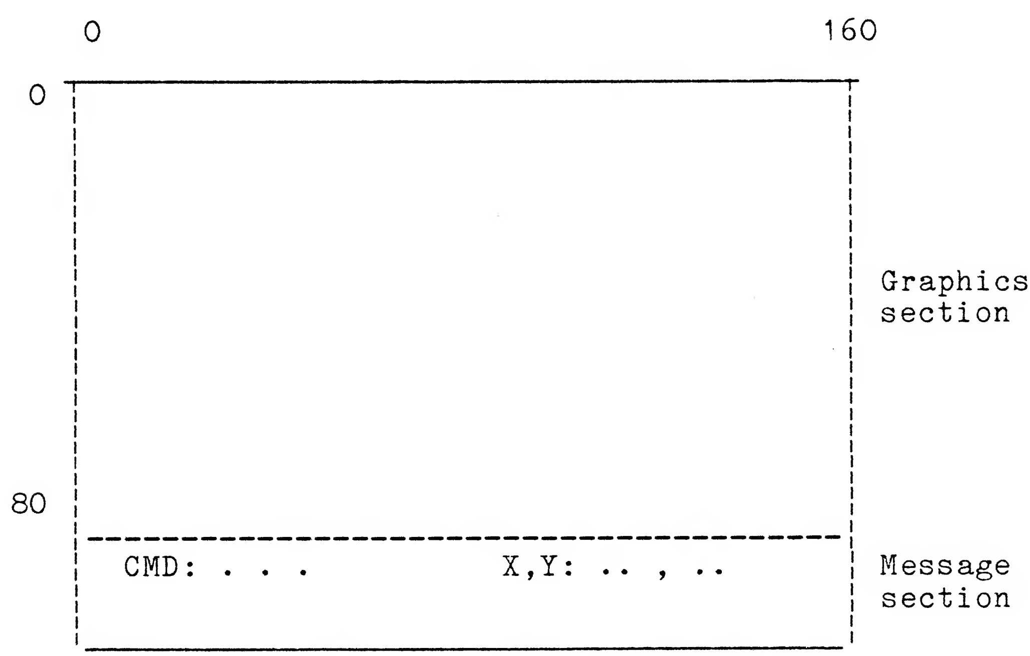


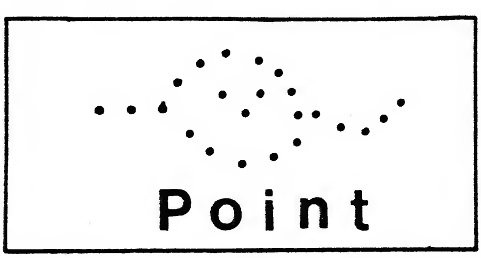
Figure VI-8 Graphics Screen

4. Using the Graphics Commands

a. Point

This command plots a point at the present cursor position.

. Position the cursor on the display screen at the desired point.
. Press the left-hand button of the KoalaPad. A point will appear at the cursor position.
. Release the left-hand button of the KoalaPad.
. Repeat the above steps, if needed.
. Press the right-hand button of the KoalaPad to return to the main menu.



b. Line

This command draws a line between two points that have been previously selected.

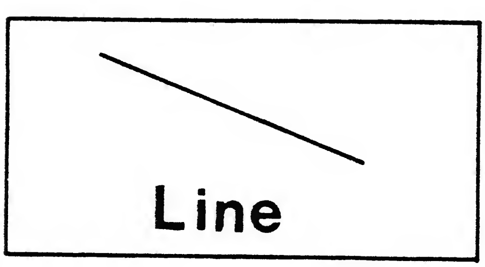
. Position the cursor on the display screen at the first point.

. Press the left-hand button of the KoalaPad to fix the first point.

. Repeat the above steps a second time to select the second point. A line will be drawn between these two points.

. Repeat the above steps to draw more lines, if needed.

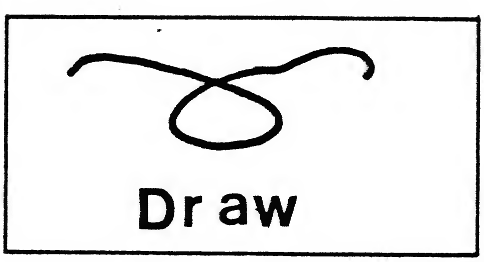
. Press the right-hand button of the KoalaPad to return to the main menu.



c. Draw

This command draws points determined by the trace of the brush (or cursor).

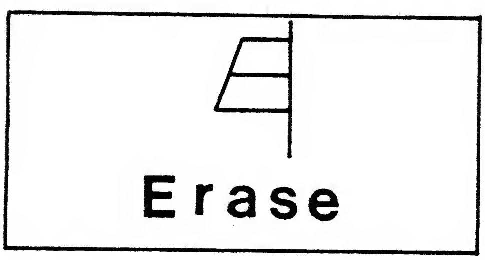
- Position the brush on the display screen at the starting location.
- Hold down the left-hand button of the KoalaPad.
- Move the brush. The points under the trace of the brush will be drawn.
- Release the left-hand button of the KoalaPad to stop drawing.
- Repeat the above steps to start drawing again at a new location, if needed.
- Press the right-hand button of the KoalaPad to return to the main menu.



d. Erase

This command erases points determined by the trace of the brush (or cursor).

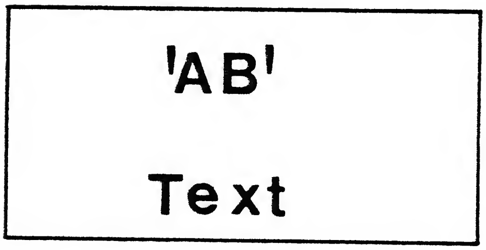
- Position the brush on the display screen at the starting location.
- Hold down the left-hand button of the KoalaPad.
- Move the brush. The points under the trace of the brush will be erased.
- Release the left-hand button of the KoalaPad to stop erasing.
- Repeat the above steps to start erasing again at a new location, if needed.
- Press the right-hand button of the KoalaPad to return to the main menu.



e. Text

This command draws a text string, which is entered from the keyboard, starting at the present cursor position. Note: the reference position for the first character is the bottom left-hand corner.

- Position the cursor on the display screen at desired point.
- Press the left-hand button of the KoalaPad to fix the reference position at the present cursor position for the first character of a text string.
- Enter a text string from the keyboard of the ATARI as instructed. The text will then be drawn on the display screen starting at the cursor position.
- Repeat the above steps to draw as many text strings as are needed.
- Press the left-hand button of the KoalaPad to return to the main menu.



f. Box

This command draws a rectangle determined by two opposite corners of that rectangle.

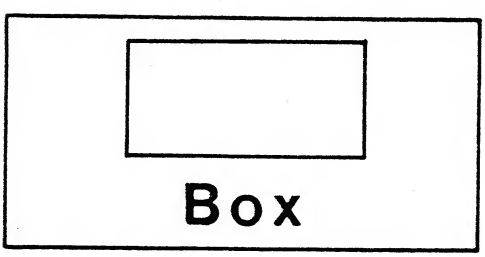
. Position the cursor on the display screen at one of the corners of the rectangle.

. Press the left-hand button of the KoalaPad to fix the cursor at this corner.

. Repeat the above steps once again to select the opposite corner of the rectangle. A rectangle will then be drawn.

. Repeat the above steps, if needed.

. Press the right-hand button of the KoalaPad to return to the main menu.



g. Fbox

This command draws a filled rectangle determined by two opposite corners of that rectangle.

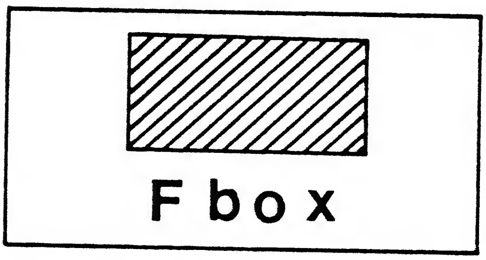
. Position the cursor on the display screen at one of the corners of the rectangle.

. Press the left-hand button of the KoalaPad to fix the cursor at this corner.

. Repeat the above steps once again to select the opposite corner of the rectangle. A filled rectangle will then be drawn.

. Repeat the above steps, if needed.

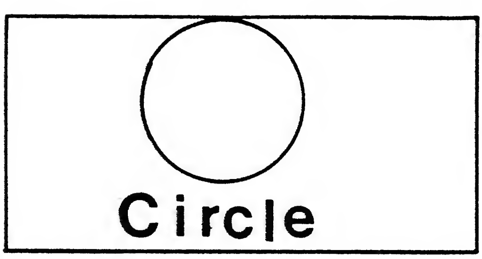
. Press the right-hand button of the KoalaPad to return to the main menu.



h. Circle

This command draws a circle with center at the first selected point and with radius determined by the distance between the first selected point and the second selected point.

- Position the cursor on the display screen at the center of a circle.
- Press the left-hand button of the KoalaPad to fix the cursor at that position.
- Move the cursor to a second point which determines the radius of the circle.
- Press the left-hand button of the KoalaPad to select that position. A circle will then be drawn.
- Repeat the above steps to draw as many circles as are needed.
- Press the right-hand button of the KoalaPad to return to the main menu.



i. Fcircle

This command draws a filled circle with center at the first selected point and with radius determined by the distance between the first selected point and the second selected point.

. Position the cursor on the display screen at the center of a circle.

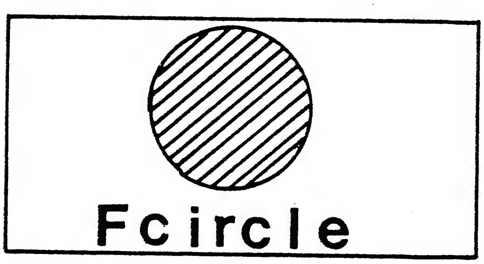
. Press the left-hand button of the KoalaPad to fix the cursor at that position.

. Move the cursor to a second point which determines the radius of the circle.

. Press the left-hand button of the KoalaPad to select that position. A filled circle will then be drawn.

. Repeat the above steps to draw as many filled circles as are needed.

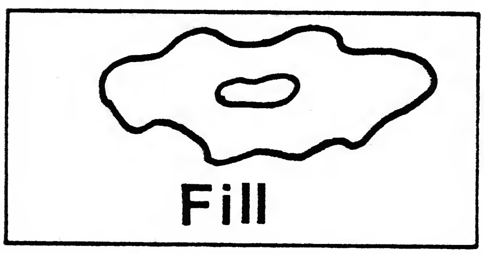
. Press the right-hand button of the KoalaPad to return to the main menu.



j. Fill

This command fills a connected closed region with the current color.

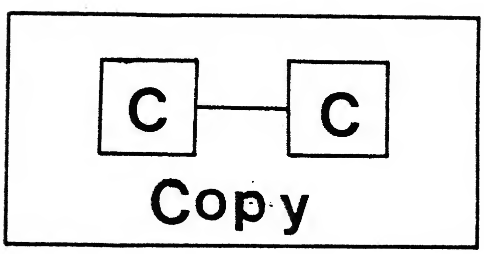
- Position the cursor on the display screen at a point within the region that is to be filled.
- Press the left-hand button of the KoalaPad to select that location. The region will then be filled with the current color.
- Repeat the above steps to fill as many regions as needed.
- Press the right-hand button of the KoalaPad to return to the main menu.



k. Copy

This command copies a rectangular region to another position.

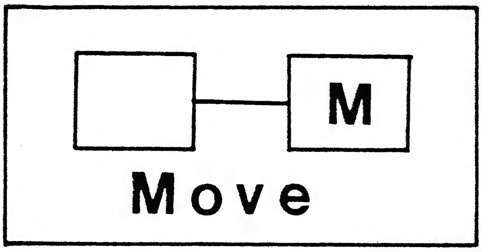
- Position the cursor on the display screen at a corner of the source rectangular region.
- Press the left-hand button of the KoalaPad to select that corner.
- Follow the same process to select the opposite corner of the source rectangular region.
- Follow in the same manner as above to select the upper left-hand corner of the destination rectangular region. The source rectangular region will then be copied to the desired destination.
- Repeat the above steps to copy as many regions as needed.
- Press the right-hand button of the KoalaPad to return to the main menu.



1. Move

This command moves a rectangular region to another position.

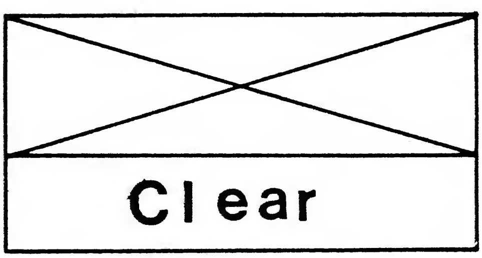
- Position the cursor on the display screen at a corner of the source rectangular region.
- Press the left-hand button of the KoalaPad to select that corner.
- Follow the same process to select the opposite corner of the source rectangular region.
- Follow in the same manner as above to select the upper left-hand corner of the destination rectangular region. The source rectangular region will then be moved to the desired destination and the source rectangular region will be filled with the background color.
- Repeat the above steps to move as many regions as needed.
- Press the right-hand button of the KoalaPad to return to the main menu.



m. Clear

This command clears the whole screen. The user must acknowledge this request by pressing the left-hand button of the KoalaPad.

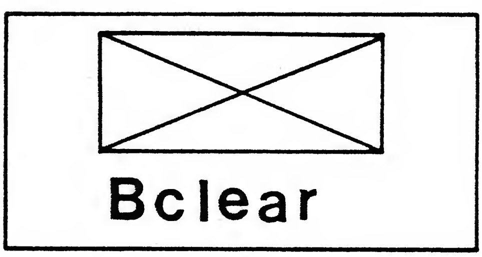
- Follow the instructions in the message section to press either the left-hand button of the KoalaPad or the right-hand button of the KoalaPad.
    - If the left-hand button of the KoalaPad has been pressed, the whole screen is cleared.
    - If the right-hand button of the KoalaPad has been pressed, this command is ignored.
- After executing this command, the user will be returned to the main menu.



n. Bclear

This command clears a rectangular (box-like) region determined by two opposite corners of that rectangular region.

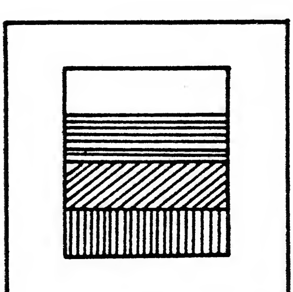
- Position the cursor on the display screen at one corner of the required rectangular region.
- Press the left-hand button of the KoalaPad to select that corner.
- Follow the same process again to select the opposite corner of that region. That region will then be filled with the background color.
- Repeat the above steps to clear as many rectangular regions as needed.
- Press the right-hand button of the KoalaPad to return to the main menu.



5. Using the Color Memu

The setcolor command displays the color menu on the left-hand side of the graphics screen. The user can select any color as one of the four possible colors (registers). See the next section on how to switch from the current color register to one of the other three possible color registers.

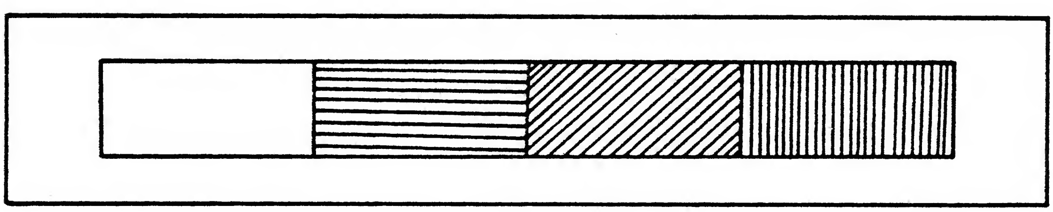
- Position the cursor within the block of the color to be selected.
- Press the left-hand button of KoalaPad to select that color. It will become the the current color for that register.
- Repeat the above steps, if needed, to select colors for the other color registers.
- To make the color menu disappear and the main menu return, either press the right-hand button of KoalaPad or press the left button of the KoalaPad while leaving the stylus off the tablet.
- While using the color menu, the usecolor and the adjust luminance commands still can be used.



6. Usecolor

The usecolor command selects a color from the four available colors.

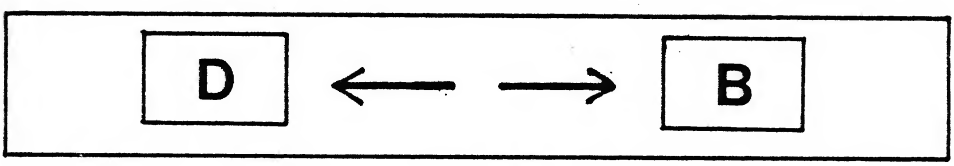
- Position the cursor within the color box that has the color to be selected as the current color.
- Press the left-hand button of the KoalaPad to select that color. A pair of bright horizontal lines will appear above and below that color box to indicate that it is the current color.
- To change the color under consideration, repeat the above steps.
- Usecolor can be executed either during the main menu command selection stage or during the color menu selection stage.



7. D<-->B

This command adjusts the luminance of the current color (Darker or Brighter).

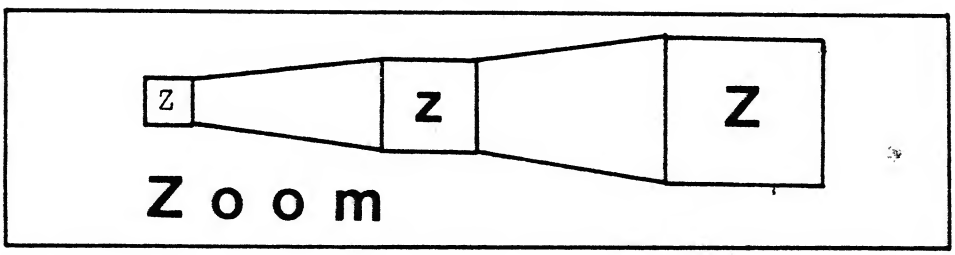
- Position the cursor within either the box which contains "D" or the box which contains "B".
- Press the left-hand button of the KoalaPad to change the current color one level in brightness. If "D" had been selected, the current color will get one level darker; while, if "B" had been selected, the current color will get one-level brighter. Beware, one level brighter than the brightest level is the darkest level. Similarly, one level darker than the darkest level is the brightest level.



8. Zoom

This command sets the zoom mode to either normal, double, or quadruple.

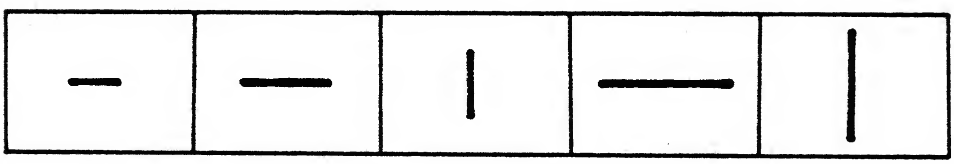
- Position the cursor within the smallest box, the medium sized box, or the largest box.
- Press the left-hand button of the the KoalaPad to set the current zoom mode to either normal, double, or quadruple.
- To select another zoom mode, follow the steps described above.



9. Setbrush

This command selects one of the five brushes for drawing or erasing.

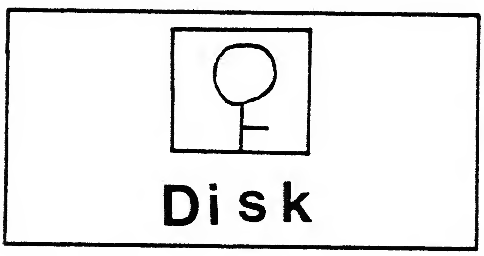
- Position the cursor in the box that contains the desired brush shape.
- Press the left-hand button of the KoalaPad to select that brush as the current brush. A pair of bright horizontal lines will appear above and below that box to indicate that it is the current brush.
- To select another brush, follow the steps described above.



10. Disk

This command displays the disk operation menu and allows the user to select one of the disk operations.

- Position the cursor at the disk operation entry.
- Press the left-hand button of the KoalaPad to select that entry.
- Follow the instructions to either enter a filename for a screen to be saved, or load a screen that has been previously saved, or delete a file from the directory, or format a diskette, or terminate the directory session.
- Select the entry "RETURN" to return to the main menu.



11. Exit

This command allows the user to exit FIGE. It displays a warning message to make sure that the current screen is saved before leaving, if that is desired. The user can select one of three options; namely, save the current screen, return to FIGE, or exit FIGE.

- Position the cursor on one of the three options.
- Press the left-hand button of the KoalaPad to select the desired option. Upon leaving FIGE, the user will be returned to FORTH.

E X I T

12. GO

This command goes ahead and executes the current command.

This command must be selected by the user whenever the current command involves any screen other than the main menu.

G O

VII. CONCLUSION

The purpose of this thesis was to design and implement in FORTH both an interactive color graphics system and an interactive color graphics editor.

FORTH Interactive Graphics System (FIGS) has been successfully developed on the ATARI 800 XL computer. It takes advantage of the advanced hardware features of the ATARI computer along with the powerful software features of FORTH. Some significant characteristics of FIGS includes:

1. FIGS is completely interactive.
   Because FORTH is an interactive environment, the user can enter specific graphics commands and see results immediately.
2. FIGS is easy to use.
   Meaningful command names have been selected. Simple parameter passing is handled through the parameter stack of the FORTH environment.
3. Commands are error tolerant.
   FIGS protects itself against accidental destruction of graphics displays by always verifying boundaries specified by the user.

4. Commands can execute in different graphics modes. Most of the graphics commands are designed to work with different graphics modes. In addition, simple mode switching commands have been provided in FIGS.
5. KoalaPad Touch Tablet is easy to interface to. Interfacing to the KoalaPad Touch Tablet is quite simple because FIGS provides a collection of interfacing commands especially designed for that purpose.
6. Graphics displays are easily saved and reloaded. With the support provided by the disk operation commands, graphics displays can be saved on disk as files and reloaded later as needed.

The FORTH Interactive Graphics Editor (FIGE) was also successfully developed on the ATARI 800 XL computer. It was designed on top of FIGS. FIGE is a menu-driven graphics editor that uses the KoalaPad as a pointing device.

There are some operations that would desirable but have not been implemented in this thesis. These operations would be slow on the ATARI computer because no efficient floating point hardware is available. One such operation is a general scaling algorithm for retangular regions of the display.

The execution speed of some commands could be improved with additional hardware support. The absent of floating-point hardware results both in using more complicated algorithms and in using more memory. The overhead due to frequent boundary checking also reduces the speed.

APPENDIX A: GLOSSARY OF WORDS FOR FIGS

word screen number high/low level

stack effects

description

Comments

X will always represent the X-coordinate.

Y will always represent the Y-coordinate.

1. Constants for FIGS

BACKGROUND 7

(-- 2C8H)

This constant contains the address of a color register used as the background color register.

BLACK 7

(-- 0)

This constant contains a color value for black.

BLUE-GREY 7

(-- A0)

This constant contains a color value for blue-grey.

BORDER 7

(-- 2C8H)

This constant contains the address of a color register used as the border color register.

CHBASE 58

(-- E000H)

This constant contains the starting address of the standard character set shape data.

COBALT-BLUE 7

(-- 60H)

This constant contains a color value for cobalt-blue.

COL-SCR 6

This constant contains the address where the OS keeps the X-coordinate of the cursor for the graphic screen.

COL-TXT 6

(-- 291H)

This constant contains the address where the OS keeps the X-coordinate of the cursor for the text window.

COLOR 6

(-- 2C4H)

This constant contains the starting address of an area where the color registers are located.

COLOR0 7

(-- 2C8H)

This constant contains the address of the color register used as color 0.

COLOR1 7

(-- 2C4H)

This constant contains the address of the color register used as color 1.

COLOR2 7

(-- 2C5H)

This constant contains the address of the color register used as color 2.

COLOR3 7

(-- 2C6H)

This constant contains the address of the color register used as color 3.

DARK-BLUE 7

(-- 90H)

This constant contains a color value for dark-blue.

DARK-GREEN 7

(-- D0H)

This constant contains a color value for dark-green.

DARK-LABENDER 7

(-- 50H)

This constant contains a color value for dark-lavender.

DARK-ORANGE 7

(-- 30H)

This constant contains a color value for dark-orange.

GRACTL 78

(-- D01DH)

This constant contains the address of the graphic control register used by the CTIA chip.

HPOSPO 78

(-- D000H)

This constant contains the address of a register which contains the horizontal position of player 0.

HSCROL 86

(-- D404H)

This constant contains the address of the register which controls the horizontal fine scrolling.

LBTN 79

(-- D404H)

This constant contains the address of a register where the OS places the value read from the left-hand button of the KoalaPad.

LMGN 6

(-- 52H)

This constant contains the address of a register used by the OS to keep the left-hand margin of the text screen.

MEDIUM-BLUE 7

(-- 80H)

This constant contains a color value for medium-blue.

NMIEN 61

(-- D40E)

This constant contains the address of the non-maskable interrupt enable register.

OLDADDR 6

(-- 5EH)

This constant contains the address of the address register used to keep the address of the cursor's position.

OLDCHR 6

(-- 5DH)

This constant contains the address of a register where the byte which contains the current pixel is saved.

OLIVE-GREEN 7

(-- BOH)

This constant contains a color value for olive-green.

ORANGE 7

(-- FOH)

This constant contains a color value for orange.

ORANGE-GREEN 7

(-- FOH)

This constant contains a color value for orange-green.

PADX 79

(-- 270H)

This constant contains the address of a register where the OS places the value read as the X-coordinate from the KoalaPad.

PADY 79

(-- 271H)

This constant contains the address of a register where the OS places the value read as the Y-coordinate from the KoalaPad.

PCLORO 7

(-- 2CO)

This constant contains the starting address where the shadow color registers for the four players are located.

RBTM 79

(-- 27D)

This constant contains the address of a register where the OS places the value read from the right-hand button of the KoalaPad.

RED 7

(-- 40H)

This constant contains a color value for red.

RED-ORANGE 7

(-- 20H)

This constant contains a color value for red-orange.

ROW-SCR 6

(-- 54)

This constant contains an address where the OS keeps the Y-coordinate of the cursor for the graphics screen.

ROW-TXT 6

(-- 290)

This constant contains an address where the OS keeps the Y-coordinate of the cursor for the text window.

RPMBS 78

(-- D407H)

This constant contains the address of the player-missile base register.

RUST 7

(-- 10)

This constant contains a color value for rust.

SCR-MEMORY 6

(-- 58H)

This constant contains the address of an address register where the starting address of a graphics screen RAM is kept.

SCR-MODE 6

(-- 57H)

This constant contains the address of a register where the current graphics mode is kept for OS use.

SCROLL-LINES 6

(-- 2BFH)

This constant contains the address of a register where the number of scroll lines is kept.

SDLSTL 6

(-- 230)

This constant contains the address of an address register where the starting address of the display list is kept.

SDMCTL 6

(-- 22FH)

This constant contains the address of the shadow DMA control register.

SETVBV 61

(-- E5CH)

This constant contains the entry address of an OS routine. This routine sets the vertical blank interrupt service routine entry vector to the current value of the system registers X(high) and Y(low).

SIZEPO 78

(-- DOO8H)

This constant contains the address of a register where the size data for the player O is kept.

STACK-PT 44

(-- D2H)

This constant contains the address of the stack pointer for an user defined stack.

TXT-MEMORY 6

(-- 294H)

This constant contains the address of an address register where the starting address for a text window screen RAM is kept.

TXT-MODE 6

(-- 293H)

This constant contains the address of a register where the current text mode is kept.

RLTRAMARINE-BLUE 7

(-- 70H)

This constant contains a color value for ultramarine-blue.

VBDEXIT 61

(-- C28AH)

THes constant contains the default entry address of the vertical blank interrupt delay service routine.

VDSLST 61

(-- 200H)

This constant contains the address of the entry vector of the display list interrupt service routine.

XTEMP 44

(-- 90)

This constant contains a address where the X register is temporarily kept during the execution of (STACK-PUSH).

WADDR 52

(-- 92)

This constant contains a address where the address of the working area is temporarily kept during the execution of LSHIFT or RSHIGT.

2. Variables and Arrays for FIGS

ADDRPO 78

(player# -- address)

This array contains the starting addresses of the four locations where ANTIC gets player shape data.

BI/PX 8

(mode# -- value)

This array contains the value for number of bits-per-pixel for each of ANTIC modes.

BITS 52

(-- address)

This variable contains the number of bits that the bytes within a horizontal line are to be shifted before they are copied to the destination.

BY/LN 8

(mode# -- value)

This array contains the values for the number of bytes per line for each of the ANTIC modes.

BYLN 44

(-- address)

This variable contains the value of the the number of bytes per line for the current mode.

CDDA 62

(-- address)

This variable contains the current value for the screen data pointer.

CDLA 62

(-- address)

This variable contains the current value of display list pointer.

CK/PX 86

(-- address)

This variable contains the value of color clocks per pixel for the current zoom mode.

CMSK 10

(-- address)

This variable contains the pixel mask value for the current pixel number.

COLMSKTB 9

(-- address)

This variable and its following 9 bytes contains the values of the color masks.

COLORMSK 10

(-- address)

This variable contains the color mask value for the current color.

CURR-COLOR 10

(-- address)

This variable contains the current color number.

CURR-MODE 10

(-- address)

This variable contains the current mode number.

CURR-PFIG 78

(-- address)

This variable contains the address of a player shape data for the current player.

CURR-PLAYER 78

(-- address)

This variable contains the current player number.

DADDR 52

(-- address)

This variable contains the starting destination address where a horizontal line is copied to or moved to.

DBYTES 52

(-- address)

This variable contains the number of bytes the destination line should have.

DDFLAG 62

(-- address)

This variable contains a flag that determines whether the screen data will be saved or not.

DL-HERE 17

(-- address)

This variable contains the next address where a display list instruction will be put at.

DLFLAG 62

(-- address)

This variable contains a flag that determines whether or not the display list will be saved.

DLPX 52

(-- address)

This variable contains the pixel number of the left most pixel of the destination horizontal line.

DOWNTRIG 44

(-- address)

This variable contains a trigger value which controls the seeds qualification for runs below the current run.

DRPX 52

(-- address)

This variable contains the pixel number of the right most pixel of the destination horizontal line.

FDA 63

(-- address)

This variable contains a fixed address where the directory sectors will be read into.

FIGPO 78

(player# -- address)

This array contains the addresses of the player shapes for the four players.

FILLFLG 38

(-- address)

This variable contains a flag which determines whether to draw a circle or a filled circle.

FIXFLAG 86

(-- address)

This variable contains a flag which determines whether a zoom window needs to be fixed.

FMSK 10

(-- address)

This variable contains both the left pixel fill mask and the right pixel fill mask for drawing a horizontal line.

HLOWPO 78

(player# -- value)

This array contains the horizontal lower limits of the coordinates for the four players.

HPSIZE 78

This array contains the player sizes for the four players.

LFMSKTB 9

(-- address)

This variable and its following 14 bytes contain the left pixel fill mask values.

NCOLORS 9

(mode# -- address)

This array contains the numbers for the colors corresponding to the 14 ANTIC modes.

NEWCOL 10

(-- address)

This variable contains the X-coordinate of the second point when drawing a line, a box, etc.

NEWROW 10

(-- address)

This variable contains the Y-coordinate of the second point when drawing a line, a box, etc.

OLDCOL 10

(-- address)

This variable contains the X-coordinate of the current cursor position.

OLDCOLMSK 44

(-- address)

While filling a region this variable contains the color mask of the old color.

OLDROW 10

(-- address)

This variable contains the Y-coordinate of the current cursor position.

PADDR 78

(-- address)

This variable contains the address of the currently used player shape data.

PDATAO 81

(-- address)

This variable and its following few bytes contain the data of a player shape. The first byte contains the length of that player shape data. The second and the third bytes contain the reference position for that player shape. The remaining bytes contain shape data.

PDATA1 80

(-- address)

This variable and its following few bytes contain the data of a player shape. The first byte contains the length of that player shape data. The second and the third bytes contain the reference position for that player shape. The remaining bytes contain shape data.

PDDA 62

(-- address)

This variable contains the starting address for the part of screen data that has not yet been saved during the execution of SAVE-DATA.

PDLA 62

(-- address)

This variable contains the starting address for the part of display list that has not yet been saved during the execution of SAVE-DATA.

PFX 79

(-- address)

This variable contains the factor value for the X-coordinate while transforming the values read from the KoalaPad to screen coordinates.

PFY 79

(-- address)

This variable contains a factor value for the Y-coordinate while transforming the values read from the KoalaPad to screen coordinate.

PHLOW 78

(-- address)

This variable contains the horizontal lower limit for the current player.

PMBASE 78

(-- address)

This variable contains the starting address of the player-missile graphics base area.

PMODE 10

(-- address)

This variable contains the current graphics mode.

PNL 78

(-- address)

This variable contains the player line size (1 or 2 scan lines).

PSIZE 78

(-- address)

This variable contains the size of base area for the current player.

PVLOW 78

(-- address)

This varible contains the vertical lower limit for the current player.

PX/BY 8

(mode# -- value)

This array contains the number of pixels per byte for each of the ANTIC modes.

PX/ML 8

(mode# -- value)

This array contains the number of pixels per mode line for each of the ANTIC modes.

PXBY 10

(-- address)

This variable contains the number of pixels per byte for the current mode.

PXML 78

(-- address)

This variable contains the number of pixels per mode line for the current mode.

PXMSKTB 9

(-- address)

This variable and its following 13 bytes contain pixel mask values for the different number of pixels per byte.

RFMSKTB 9

(-- address)

This variable and its following 13 bytes contain the left pixel fill mask values for the different number of pixels per byte.

RPXPO 78

(player# -- value)

This array contains the horizontal values for the reference positions of the player shapes for the four players.

RPYPO 78

(player# -- value)

This array contains the vertical values of the reference positions for the player shapes for the four players.

SADDR 52

(-- address)

This variable contains the starting address of the source where a horizontal line is copyed or moved from.

SDXA 52

(-- address)

This variable contains the number of bytes in the source horizontal line.

SL/ML 8

(mode# -- value)

This array contains the number of scan lines per mode line for each of the 14 ANTIC modes.

SLML 78

(-- address)

This variable contains the number of scan lines per mode line for the current mode.

STACK-BTM 44

(-- address)

This variable contains the address of the bottom of the user defined stack.

TNEWCOL 10

(-- address)

This variable temporarily contains the X-coordinate of the second point during the execution of CLINE.

TNEWROW 10

(-- address)

This variable temporarily contains the Y-coordinate of the second point during the execution of CLINE.

TOLDCOL 10

(-- address)

This variable temporarily contains the X-coordinate of the first point during the execution of CLINE.

TOLDROW 10

(-- address)

This variable temporarily contains the Y-coordinate of the first point during the execution of CLINE.

TSDMCTL 10

(-- address)

This variable temporarily contains the value of SDMCTL.

TXO 10

(-- address)

This variable temporarily contains the X-offset value.

TXMAX 10

(-- address)

This variable contains the largest X-boundary value for the terminal screen window.

TXMIN 10

(-- address)

This variable contains the smallest X-boundary value for the terminal screen window.

TYO 10

(-- address)

This variable temporarily contains the Y-offset value.

TXMAX 10

(-- address)

This variable contains the largest Y-boundary value for the terminal screen window.

TYMIN 10

(-- address)

This variable contains the smallest Y-boundary value for the terminal screen window.

UPTRIG 44

(-- address)

This variable contains a trigger value which controls the seeds qualification for the runs above the current run.

VL 29

(-- address)

This variable contains a value that indicates whether the value of ¦X2-X1¦ is less than the value of ¦Y2-Y1¦; if so, VL contains 1.

VLOWPO 78

(player# -- value)

This array contains the vertical lower limits for the coordinates of the four players.

VPOSPO 78

(player# -- value)

This array contains the Y-coordinates of the current positions for the four players.

WORKING 52

(-- address)

This variable contains the starting address of a working area for moving or copying a horizontal line.

XO 10

(-- address)

This variable contains the current X-offset value.

XMAX 10

(-- address)

This variable contains the largest X-boundary value for the logical window.

XMIN 10

(-- address)

This variable contains the smallest X-boundary value for the logical window.

XSIGN 29

(-- address)

This variable contains:

-1, if X2 is less than X1

0, if X2 equals to X1

1, if X2 is greater than X1.

It is set by DIRECTION.

XYSIGN 29

(-- address)

This variable contains the vesulting from the operation value XSIGN * YSIGN.

YO 10

(-- address)

This variable contains the current Y-offset value.

YMAX 10

(-- address)

This variable contains the largest Y-boundary value for the logical window.

YMIN 10

(-- address)

This variable contains the smalleat Y-boundary value for the logical window.

YSIGN 29

(-- address)

This variable contains:

-1, if Y2 is less than Y1

0, if Y2 equals to Y1

1, if Y2 is greater than Y1.

It is set by DIRECTION.

ZDL-ADDR 86

(-- address)

This variable contains the starting address of the display list for the current zoom mode.

ZFLAG 86

(-- address)

This variable contains a flag value that indicates if the zoom mode is active or not. A value 1 means active.

ZMLX 86

(-- address)

This variable contains the width in X-direction of the zoom window for the current zoom mode.

ZMLY 86

(-- address)

This variable contains the width in Y-direction of the zoom window for the current zoom mode.

ZMODE 86

(-- address)

This variable contains the current zoom mode.

ZXO 86

(-- address)

This variable contains the X-coordinate of the upper left-hand corner for the current zoom window.

ZXHIG 86

(-- address)

This variable contains the largest X-limit value that ZXO can be.

ZXLOW 86

(-- address)

This variable contains the smallest X-limit value that ZXO can be.

ZYO 86

(-- address)

This variable contains the Y-coordinate of the upper left-hand corner of the current zoom window.

ZYHIG 86

(-- address)

This variable contains the largest Y-limit value that ZYO can be.

ZYLOW 86

(-- address)

This variable contains the smallest Y-limit value that ZYO can be.

3. High-level and Low-level words for FIGS

!ZOOM-DLS 87 H

(DL-addr screen-addr zoom-mode base-mode lines --)

This word creates a given number of mode lines in a display list starting at the given address.

(=OLDCOLOR?) 45 L (ASM ONLY)

(address -- flag)

This word is an assembly language subroutine which is called by other low level code words to see if the pixel at the given address has the same value as that stored in the variable OLDCOLMSK; if so, the flag is set to true.

(DELETE) 68 H

(address flag --)

If the flag is true, this word deletes a file directory entry whose information is at the given address; otherwise, it displays the message: "Filename Not Found!".

(GLOAD) 76 H

(address flag --)

If the flag is true, this word reloads the screen data and/or the display list from a file whose directory information is at the given address.

(GSAVE) 74 H

(address flag --)

If the flag is true, this word saves the screen data and/or the display list in a file whose directory information is at the given address.

(PRINT) 60 H

(X Y --)

This word draws the text string, which is already in the dictionary, on the screen surface starting at the given X-Y coordinates.

(STACK-PUSH) 46 L (ASM ONLY)

(X Y --)

This word is an assembly language subroutine which is called by other low level words to push the given X-Y coordinates on top of the user defined stack.

+OFFSET 12 H

(X Y -- X-offset Y-offset)

This word adds the current offset values to the given X-Y coordinates.

<FILL 49 H

(X Y -- address pixel# X Y)

This word scans each pixel of a run from the given starting point towards the left to find the left boundary position of that run and puts that address, the pixel number, and the X-Y coordinates of that left boundary position on top of the parameter stack.

=OLDCOLOR? 45 L

(address -- flag)

This word determines if the pixel at the given address has the same color value as that stored in the variable OLDCOLMSK; if so, the flag is set to true.

>CREG 13 H

(color# -- address)

This word converts the given color number to the address of the corresponding color register.

>INCODE 58 H

(ATASCII-code -- incode)

This word converts the given ATASCII-code to the Incode for the ATARI computer.

>PXY 84 H

(X Y -- X-player Y-player)

This word transforms the given X-Y coordinates to player X-Y coordinates.

>SCRXY 80 H

(X-Pad Y-Pad -- X Y)

This word transforms the given coordinates read from the KoalaPad Touch Tablet to screen X-Y coordinates.

>TEMP 29 H

(X-1 Y-1 X-2 Y-2 --)

This word stores the given two points in temporary variables with the X-Y coordinates exchanged when the value in the FORTH variable VL is 1.

>ZOOMXY 88 H

(X Y -- X-zoom Y-zoom)

This word moves the zoom window by transforming the given X-Y coordinates to the zoomed X-Y coordinates.

ABOVE? 47 L

(address X Y --)

This word examines the value in the pixel position above the given pixel position to see if this pixel position should be the starting point for a new run; if so, this pixel position is pushed on top of the stack.

BCLEAR 43 H

(X-1 Y-1 X-2 Y-2 --)

This word clears a retangular region determined by the given two opposite-corner positions by filling that region with the background color.

BEGIN-DL 17 H

(DL-address -- DL-address)

This word puts 3 bytes of 70H into memory, starting at the address that is on top of the parameter stack.

BELOW? 48 L

(address X Y --)

This word examines the value in the pixel position below the given pixel position to see if this pixel position should be the starting point for a new run; if so, this pixel position is pushed on top of the stack.

BFILL 51 H

(X Y --)

This command fills a connected region with the current color starting at the given starting point. If the pixel at the starting point already has the same value as the current color number, this command terminates immediately.

BOUNDARY? 11 H

(X Y -- flag)

This commands checks whether a given pair of X-Y coordinates are within the logical window boundary.

BOX 41 H

(X-1 Y-1 X-2 Y-2 --)

This command draws a rectangular region determined by two opposite-corner points for that rectangular region.

BRIGHTER 13 H

(color# --)

This command increases the luminance value in the given color register by two so that the color becomes 1 level brighter.

CHARACTER 59 H

(X Y code --)

This command draws on the display starting at the given position the character whose shape is given by the corresponding code.

CIRCLE 41 H

(X Y radius --)

This command draws a circle with center at the given point with the given radius.

CIRCLE-FILL 38 H

(X Y --)

This command draws four horizontal line segments between some of the eight symmetric points.

CIRCLE-PTS 39 H

(X Y --)

This command plots the eight symmetric points for the given point on the circle.

CIRCLE-PUT 39 H

(X Y --)

This command checks the value of the fill-flag (FILLFLG) to determine whether to call CIRCLE-FILL or CIRCLE-PTS.

CLEAR 43 H

(--)

This command clears the whole screen.

CLINE 37 H

(X-coordinate-1 Y-coordinate-1 X-coordinate-2 Y-coordinate-2 --)

This command is called by DRAWLINE to draw a line between the given two points.

DARKER 13 H

(color# --)

This command decreases the luminance value in the given color register by two so that the color becomes 1 level darker.

DELETE 69 H

(--)

This command deletes the given filename from the file directory and reassigns the used data sectors to the free sector list.

DIR 77 H

(--)

This command lists the filenames for the files that exist on the files diskette.

DIR? 34 H

(value-1 value-2 -- direction)

This words compares the given two values. If the first value is smaller than the second value, then the value assigned to direction is -1; otherwise, the value assigned is 1.

DIRECTION 35 H

(X-1 Y-1 X-2 Y-2 -- X-direction Y-direction)

This command evaluates the direction values for a vector from the first point to the second point.

DLI-OFF 61 H

(--)

This command disables the display list interrupt routine.

DLI-ON 61 H

(--)

This command enables the display list interrupt routine.

DOUBLE-LINE 82 H

(--)

This command sets the player line size option to double-line.

DOUBLE-SIZE 81 H

(player# --)

This command sets the player size to double-size (8 color clock width).

DRAWCIRCLE 40 H

(X Y radius --)

This command calculates the coordinates of the points on a circle and determined by the value of the fill-flag (FILLFLG) calls CIRCLE-PUT to either draw a circle or draw a filled circle.

DRAWLINE 37 H

(--)

This command draws a line between two points which are maintained in the FORTH variables OLDCOL, OLDROW, NEWCOL, and NEWROW.

DRAWTO 38 H

(X Y --)

This command draws a line from the current cursor position to the given point.

END-DL 17 H

(address --)

This command concludes the display list with a jump instruction that jumps to the given display list address.

FBOX 43 H

(X-1 Y-1 X-2 Y-2 --)

This command draws a filled rectangular region determined by the two given opposite-corner points for that retangular region.

FCIRCLE 41 H

(X Y radius --)

This command draws a filled circle centered at the given point with the given radius.

FILBOX 42 H

(--)

This command draws a filled rectangular region determined by two opposite-corner points which are maintained in the FORTH variables OLDCOL, OLDROW, NEWCOL, and NEWROW.

FILL> 50 H

(address pixel# X Y --)

This word is called by BFILL. It fills the current run with the current color and checks the rows above and below the current run to determine whether some more runs are required. If more runs are required, FILL> puts the seeds of these runs on the stack.

FIND-FD 67 H

(address-1 -- address-2 flag)

This command attempts to find the address of the directory entry with the filename that is kept at the given address; if the filename is found, its address along with a flag with value 1 are returned; otherwise, only a flag with value 0 is returned.

FIND-FN 67 H

(-- address flag)

This command obtains a filename from the current input stream and calls FIND-FD to find the address of a directory entry with that name.

FIND-EMPTY 68 H

(-- address flag)

This command determines an entry for a new directory item.

FLUSHDL 87 H

(address pixel# --)

This command moves the zoom window to a location determined by the given point. This is done by changing the load memory address for each mode line in the display list and applying fine scrolling.

FORMAT 65 H

(--)

This command formats a disk with the format necessary for this simple disk file management system.

FORMAT-FDD 65 H

(--)

This word is called by FORMAT to format the appropriate sectors for a directory.

FORMAT-FDS 65 H

(--)

This word is called by FORMAT to format the appropriate sectors for the free sector list.

FORMAT-MFD 64 H

(--)

This word is called by FORMAT to format the sector for the master directory.

GET-MFD 64 H

(--)

This command reads the master directory sector into memory.

GETCOLMSK 12 H

(color# -- mask-value)

This command gets the corresponding mask-value for the given color from the color mask table.

GLOAD 76 H

(--)

This command reloads the screen data and/or the display list from the given file. The filename is obtained from the input stream.

GSAVE 74 H

(--)

This command saves the display list and/or the screen data as a file on disk. The filename is obtained from the input stream.

HCALINC 29 H

(-- inc1 inc2 diffence end begin)

This command calculates the incremental values for HDRAWPTS.

HCLIP 35 H

(X-1 Y-1 X-2 Y-2 -- X-1-clipped Y-1 X-2-clipped Y-2)

This command clips the X-coordinates for the given two points.

HDRAW 33 L

(address bytes --)

This word draws a horizontal line starting at the given address with the length determined by the proper number of pixels within the bytes.

HDRAWPTS 30 H

(inc1 inc2 diffence end begin --)

This word applys Bresenham's Line Algorithm to draw a line by using the incremental values.

HLINE 36 H

(X-1 Y-1 X-2 Y-2 --)

This word draws a horizontal line between the two given points.

HPOS 42 H

(X-1 Y-1 X-2 Y-2 -- address bytes)

This word converts the coordinates for the two given points to a starting address and the proper number of pixels within the bytes between them. It also sets the masks for the two end bytes.

INCADR 32 L (ASM ONLY)

(--)

This word is an assembly language routine which is called by HDRAW to increment or decrement, determined by the value in FORTH variable XSIGN, the address in the FORTH variable OLDADR.

LINE 38 H

(X-1 Y-1 X-2 Y-2 --)

This command draws a line between the given two points.

LMOST 54 L

(working-addr left-byte-addr --)

This word copies some pixels from the left-most byte of the source line to the corresponding byte in the working area.

LMOVE 55 H

(--)

This word copies a horizontal line from the source retangular region to the destination.

LOAD-DATA 76 H

(--)

This word reloads the screen data and/or display list from the data sectors of a file.

LOAD-ENV 73 H

(--)

This word reloads the saved environment information from a file.

LOCATE 27 H

(X Y --)

This command moves the cursor position to the given position.

LSHIFT 52 L

(address bytes bits --)

This word shifts the bits of some contiguous bytes at the given address by the a given number of bits.

LUMINACE+! 13 H

(color# increment --)

This word adjusts the luminance value of the given color by the given incremental value.

MATCH? 66 H

(address-1 address-2 -- flag)

This word compares two character strings starting at the two given addresses; if the two strings are identical, a flag with value 1 is returned.

MODE-LINE 17 H

(address mode# lines --)

This word creates a part of a display list with the given number of lines with the given mode number and with a load memory scan option.

MODE0 18 H

(--)

This command switches the present graphics mode to graphics mode 0.

MODE1 19 H

(--)

This command switches the present graphics mode to graphics mode 1.

MODE2 20 H

(--)

This command switches the present graphics mode to graphics mode 2.

MODE3 21 H

(--)

This command switches the present graphics mode to graphics mode 3.

MODE4 22 H

(--)

This command switches the present graphics mode to graphics mode 4.

MODE5 23 H

(--)

This command switches the present graphics mode to graphics mode 5.

MODE6 24 H

(--)

This command switches the present graphics mode to graphics mode 6.

MODE7 25 H

(--)

This command switches the present graphics mode to graphics mode 7.

MODE8 26 H

(--)

This command switches the present graphics mode to graphics mode 8.

OFFSET! 12 H

(X-value Y-value --)

This command stores new offset values in the FORTH variables XO and YO.

OFFSET>T 12 H

(--)

This command temporarily saves the current offset values.

OFFSET@ 12 H

(-- X-value Y-value)

This command puts the current offset values on top of the parameter stack.

OUTCODE 34 H

(X Y -- outcode)

This command evaluates the outcode value for the given point by comparing that point with the logical screen boundaries.

PADXY 79 L

(-- X Y)

This command reads the X-Y coordinates of the stylus and puts them on top of the parameter stack.

PLAYER-CLEAR 82 H

(--)

This command clears the player shape data for the current player.

PLAYER-OFF 82 H

(--)

This command turns off the player features.

PLAYER-ON 82 H

(--)

This command turns on the player features.

PLOT 28 H

(X Y --)

This command plots the given point on the display.

POSITION 27 H

(X Y -- address pixel#)

This command transforms the given X-Y coordinates to a memory address and a pixel number.

PREPARE 56 H

(X-1-source Y-1-source X-2-source Y-2-source X-dest Y-dest -- Y-dest-end Y-dest-begin)

This word calculates required data for LMOVE and puts it in the corresponding FORTH variables.

PRINT 59 H

(X Y address length --)

This word draws starting at the given position a text string that is located at the given address.

PUT-FDD 64 H

(--)

This command writes a directory sector to the disk.

PUT-FDS 64 H

(--)

This command writes a data sector to the disk.

PUT-MFD 64 H

(--)

This command writes the master directory sector to the disk.

PX! 31 L (ASM ONLY)

(--)

This word is an of assembly language routine which is called by the low-level word HDRAW to put a pixel value in the screen RAM.

PXPUT 28 L

(address --)

This word puts a pixel value at the given address.

QUAD-SIZE 81 H

(player# --)

This command sets the player size to quadruple-size (32 color clock width).

RCOPY 57 H

(X-1-source Y-1-source X-2-source Y-2-source X-dest Y-dest --)

This command copies the contents of a rectangular region to another place within the same screen. The contents of the source region remain unchanged after the execution of this command.

READ 64 H

(address sector# --)

This command reads the contents of the given sector into memory starting at the given address.

READPX 44 H

(X Y -- pixel-value)

This command reads the pixel value for the given point.

RELEASE 66 H

(--)

This command reassigns a used sector to the free sector list of the disk file management system.

REQUEST 69 H

(--)

This command requests a sector from the free sector list of the disk file management system.

RESETVBD 61 L

(--)

This command resets the vertical blank interrupt delay phase (VBD) entry vector to its default value.

RESORT 27 L

(X-1 Y-1 X-2 Y-2 -- X-upper-left Y-upper-left X-bottom-right Y-bottom-right)

This command rearranges the given coordinates so that the first pair become the coordinates of the upper left-hand point and the second pair become the coordinates of the bottom right-hand point.

RMOST 54 L

(--)

This word copies the appropriate pixels that are in the right most byte of the source line to the corresponding byte in the working area.

RMOVE 57 H

(X-1-source Y-1-source X-2-source Y-2-source X-dest Y-dest --)

This command copies the contents of a retangular region to another location within the same screen and fills the source rectangular region with the background color.

RSHIFT 53 L

(address bytes bits --)

This word shifts to the right a given number of bits all the bits within a contiguous area of bytes starting at the given address.

SAVE-ALL 75 H

(--)

If the filename is unique, this command saves both the screen data and display list in that file.

SAVE-DD 75 H

(--)

If the filename is unique, this command saves the screen data in that file.

SAVE-DL 75 H

(--)

If the filename is unique, this command saves the display list in that file.

SCR-LOCATE 13 H

(X-coordinate Y-coordinate --)

This command sets the graphics screen cursor position to the point with the given X-Y coordinates.

SCREEN-OFF 11 H

(--)

This command turns off ANTIC by storing 0 in SDMCTL (location 22FH). It also saves the old value from this control register in a temporary location for possible restoring by the SCREEN-ON command.

SCREEN-ON 11 H

(--)

This command turns on ANTIC and restores the DMA functions that had been previously working before the SCREEN-OFF command turned them off.

SET-DEFAULTS 16 H

(--)

This word sets the default values for the graphics mode environment.

SETBYLN 44 H

(--)

This word sets the FORTH variable BYLN to the number of bytes needed for a mode line in the current mode.

SETCMSK 27 H

(pixel# --)

This word gets a pixel mask for the given pixel number from the pixel mask table (PXMSKTB) and stores it in the FORTH variable CMSK.

SETDLIV 61 H

(address --)

This command stores the given address in the display list interrupt handler vector which is maintained at location 200H.

SETMSK 31 L

(pixel#1 pixel#2 --)

This word sets the pixel masks for the two end bytes of a horizontal line.

SETPF 80 H

(--)

This word calculates the two pad-screen transformation factors for the current mode and stores them in the FORTH variables PFX and PFY.

SETPFIG 83 H

(address X-reference Y-reference player# --)

This command sets the address where a piece of player shape data is kept and sets the reference position for the player shape for the given player.

SETPLAYERS 83 H

(--)

This command sets the default values for the player graphics environment.

SETPLINE 81 H

(code lines length address --)

This word is called by SINGLE-LINE and DOUBLE-LINE to set the required data for them.

SETPSIZE 81 H

(player# code --)

This word sets the given code of specific player size for the given player.

SETSTACK 45 H

(address --)

This command creates a stack with its bottom at the given address.

SETBVD 61 L

(address --)

This command sets the vertical blank interrupt delay phase (VBD) entry vector to the given address.

SINGLE-LINE 82 H

(--)

This command sets the player line size option to single-line (one scan line).

SINGLE-SIZE 81 H

(player# --)

This command sets the player size for the given player to single-size (8 color clock width).

STACK-EMPTY? 46 H

(-- flag)

This command checks whether the user defined stack is empty; if so, the returned flag value is 1.

STACK-POP 46 H

(-- value-1 value-2)

This command pops the top two elements of the user defined stack and leaves them on top of the parameter stack.

STACK-PUSH 47 L

(value-1 value-2 --)

This command pushes the two given values on top of the user defined stack.

STORE-DATA 71 H

(address --)

Depending on the value of DDFLAG and DLFLAG, this word is called by (GSAVE) to save a display list and/or screen data in a disk file.

STORE-DD 70 H

(--)

This word stores part of the screen data in a disk file.

STORE-DL 70 H

(--)

This word stores part of the display list in a disk file.

STORE-ENV 73 H

(--)

This word stores the the graphics mode environment.

STRING" 60 H

(X Y --)

This word is an immediate word. At compile time, it compiles (PRINT) and the string from the input stream into the dictionary. At execution-time, it executes (PRINT) to draw the compiled string on the display.

T>OFFSET 12 H

(--)

This command restores the current offset values to the values previously saved in TXO and TYO.

TEXT 60 H

(X-coordinate Y-coordinate --)

This command draws a text string, entered interactively by user, on the display starting at the given point.

TW>W 11 H

(--)

This command restores the current logical window boundaries to the current terminal screen boundaries.

TWINDOW 11 H

(X-min Y-min X-max Y-max --)

This command sets the current logical screen boundaries to the given values.

TXT-LOCATE 13 H

(X Y --)

This command sets the text window cursor position to the given X-Y coordinates.

USECOLOR 12 H

(color# --)

This command sets the current color number to the given color number.

USEPLAYER 84 H

(player# --)

This command sets the current player to the given player.

VCLIP 36 H

(X1 Y-1 X-2 Y-2 -- X-1 Y-1-clipped X-2 Y-2-clipped)

This command clips the Y-coordinates for the given two points.

VDRAW 32 L

(length bytes/line --)

This word draws a vertical line with the given length starting at the address kept in OLDADR.

VLINE 36 H

(X-1 Y-1 X-2 Y-2 --)

This word draws a vertical line between the two given points.

WCLEAR 43 H

(--)

This command fills the whole logical window with the background color.

WINDOW 11 H

(X-min Y-min X-max Y-max --)

This command sets the current logical window boundaries to the given values.

WRITE 64 H

(address sector# --)

This command writes the block of data at the given address to the given disk sector.

WTCRS 28 H

(--)

This command puts a pixel value on the display at the position kept in the FORTH variables OLDCOL and OLDROW.

XCLIP 35 H

(X-coordinate -- clipped-X)

This command clips the given X-coordinate to the logical boundaries.

YCLIP 35 H

(Y -- clipped-y)

This command clips the given Y-coordinate to the logical boundaries.

ZPLAYER 88 H

(X Y --)

This command displays, depending on the value of the zoom flag (ZFLAG), a player either on a zoom mode screen or on a normal screen at the given position.

APPENDIX B: SOURCE LISTING FOR FIGS

```
SCR #3
( GRAPHICS -- PREDEFINED TOOLS )
HEX
: CARRAY  ( N -- )
    <BUILDS 1+ ALLOT DOES> +
;
: SARRAY  ( N LEN -- )
    <BUILDS DUP C, SWAP 1+ * ALLOT
    DOES> DUP C@ >R SWAP R * + 1+ R>
;
: SARRAY!  ( ADDR1 ADDR2 LEN -- )
    2DUP BL FILL >R SWAP COUNT ROT SWAP R> MIN CMOVE
;
: SARRAY?  ( ADDR LEN -- )
    TYPE
;
-->
```

```
SCR #4
( GRAPHICS -- PREDEFINED TOOLS )
: NOT 0= ;  ( F -- NOT-F )
: U> SWAP U< ;  ( N1 N2 -- F )

CODE J  ( -- J )
  XSAVE STX, TSX, 105 ,X LDA, PHA,
  106 ,X LDA, XSAVE LDX, PUSH JMP,

CODE 2R   (  -- X Y )
    DEX, DEX, DEX, DEX, PLA, 2 ,X STA,
    PLA, 3 ,X STA, PLA, 0 ,X STA, PLA,
    1 ,X STA, PHA, 0 ,X LDA, PHA,
    3 ,X LDA, PHA, 2 ,X LDA, PHA,
    NEXT JMP,
FORTH
-->
```

SCR #5

```
( GRAPHICS -- RECORD 3/27/85 )
: RECORD    ( ADDR --   )
    VARIABLE 0 , 0 LATEST PFA
;
: %       ( COUNT PFA -- COUNT PFA )
    <BUILDS DUP , OVER ,
    DOES> DUP @ @ SWAP 2 + @ +
;
: BYTES  ROT + SWAP ;  ( COUNT PFA N -- COUNT+N PFA )
: END-RECORD  2 + ! ;  ( COUNT PFA --   )
: LEN'  (  -- LENGTH  )
    -FIND
    IF DROP 2 + @
    ELSE ." UNDEFINED RECORD NAME "
    ENDIF
;  -->
```

SCR #6

```
(  SYSTEM CONSTANTS  )
VOCABULARY GRAPHICS GRAPHICS DEFINITIONS
HEX
52 CONSTANT LMGN          53 CONSTANT RMGN
54  CONSTANT ROW-SCR      55  CONSTANT COL-SCR
57  CONSTANT SCR-MODE     58  CONSTANT SCR-MEMORY
5D CONSTANT OLDCHR        5E CONSTANT OLDADR
2C0 CONSTANT PCOLR0       2C4 CONSTANT COLOR
22F CONSTANT SDMCTL       230 CONSTANT SDLSTL
290 CONSTANT ROW-TXT      291 CONSTANT COL-TXT
293 CONSTANT TXT-MODE     294 CONSTANT TXT-MEMORY
2BF CONSTANT SCROLL-LINES

-->
```

SCR #7

```
( GRAPHIC -- REGISTERS )
0 CONSTANT BLACK  10 CONSTANT RUST
20 CONSTANT RED-ORANGE  30 CONSTANT DARK-ORANGE
40 CONSTANT RED  50 CONSTANT DARK-LAVENDER
60 CONSTANT COBALT-BLUE  70 CONSTANT ULTRAMARINE-BLUE
80 CONSTANT MEDIUM-BLUE  90 CONSTANT DARK-BLUE
A0 CONSTANT BLUE-GREY  B0 CONSTANT OLIVE-GREEN
C0 CONSTANT MEDIUM-GREEN  D0 CONSTANT DARK-GREEN
E0 CONSTANT ORANGE-GREEN  F0 CONSTANT ORANGE
2C8 CONSTANT BORDER  2C8 CONSTANT BACKGROUND
2C8 CONSTANT COLOR0  2C4 CONSTANT COLOR1
2C5 CONSTANT COLOR2  2C6 CONSTANT COLOR3



-->
```

SCR #8

```
( GRAPHIC -- CONT. #2 )

F CARRAY SL/ML -E ALLOT
  0A08 , 1008 , 1008 , 0408 , 0204 , 0201 , 0101 ,
F CARRAY BY/LN -E ALLOT
  2828 , 2828 , 1414 , 0A0A , 1414 , 2814 , 2828 ,
F ARRAY PX/ML -1C ALLOT
  28 , 28 , 28 , 28 , 14 , 14 , 28 , 50 , 50 ,
  A0 , A0 , A0 , A0 , 140 ,
F CARRAY PX/BY -E ALLOT
  0101 , 0101 , 0101 , 0804 , 0804 , 0408 , 0804 ,
F CARRAY BI/PX -E ALLOT
  0808 , 0808 , 0808 , 0102 , 0102 , 0201 , 0102 ,
F CARRAY NCOLORS -E ALLOT
  0202 , 0404 , 0505 , 0204 , 0204 , 0402 , 0204 ,
-->
```

SCR #9

```
( GRAPHICS -- TABLES CONT. )
0 VARIABLE PXMSKTB -2 ALLOT
    FF C, F0 C, 0F C, C0 C, 30 C, 0C C, 03 C, 80 C,
    40 C, 20 C, 10 C, 08 C, 04 C, 02 C, 01 C,
0 VARIABLE COLMSKTB -2 ALLOT
    00 C, FF C, 00 C, 55 C, AA C, FF C, FF C, FF C,
    FF C, FF C,
0 VARIABLE LFMSKTB -2 ALLOT
    FF C, F0 C, FF C, C0 C, F0 C, FC C, FF C, 80 C,
    C0 C, E0 C, F0 C, F8 C, FC C, FE C, FF C,
0 VARIABLE RFMSKTB -2 ALLOT
    FF C, FF C, 0F C, FF C, 3F C, 0F C, 03 C, FF C,
    7F C, 3F C, 1F C, 0F C, 07 C, 03 C, 01 C,
-->
```

SCR #10

```
( GRAPHICS -- VARIABLES )
0 VARIABLE OLDCOL    0 VARIABLE OLDROW
0 VARIABLE NEWCOL    0 VARIABLE NEWROW
0 VARIABLE TOLDCOL   0 VARIABLE TOLDROW
0 VARIABLE TNEWCOL   0 VARIABLE TNEWROW
0 VARIABLE XMIN      0 VARIABLE YMIN
0 VARIABLE XMAX      0 VARIABLE YMAX
0 VARIABLE TXMIN     0 VARIABLE TYMIN
0 VARIABLE TXMAX     0 VARIABLE TYMAX
0 VARIABLE X0        0 VARIABLE Y0
0 VARIABLE TX0       0 VARIABLE TY0
0 VARIABLE CMSK      0 VARIABLE FMSK
0 VARIABLE PXBY      0 VARIABLE COLORMSK
0 VARIABLE CURR-MODE 0 VARIABLE CURR-COLOR
0 VARIABLE TSDMCTL   0 VARIABLE PMODE
-->
```

```
SCR #11
( GRAPHICS -- BASIC WORDS )

: SCREEN-OFF SDMCTL C@ TSDMCTL C! 0 SDMCTL C! ;  ( -- )
: SCREEN-ON  TSDMCTL C@ SDMCTL C! ;  ( -- )
: WINDOW  ( XMIN YMIN XMAX YMAX -- )
    YMAX ! XMAX ! YMIN ! XMIN ! ;
: TWINDOW  ( TXMIN TYMIN TXMAX TYMAX -- )
    TYMAX ! TXMAX ! TYMIN ! TXMIN ! ;
: TW>W  ( -- )
    TXMIN @ TYMIN @ TXMAX @ TYMAX @ WINDOW ;
: BOUNDARY?  ( X Y -- F )
    DUP YMIN @ < SWAP YMAX @ > OR SWAP
    DUP XMIN @ < SWAP XMAX @ > OR OR NOT
;

-->
```

```
SCR #12
( GRAPHICS -- BASIC WORDS )

: OFFSET!  YO ! XO ! ;  ( XO YO -- )
: +OFFSET  ( X Y -- X+XO Y+YO )
    YO @ + SWAP XO @ + SWAP
;
: OFFSET@  XO @ YO @ ;  ( -- XO YO )
: OFFSET>T  OFFSET@ TYO ! TXO ! ;  ( -- )
: T>OFFSET  TXO @ TYO @ OFFSET! ;  ( -- )
: GETCOLMSK  ( COLOR# -- COLMSK )
    CURR-MODE C@ NCOLORS C@ DUP 5 <
    IF 2 - ENDIF + COLMSKTB + C@ ;
: USECOLOR  ( COLOR# -- )
    DUP CURR-COLOR C! GETCOLMSK COLORMSK C! ;

-->
```

```
SCR #13
( GRAPHICS -- DEMO )

: >CREG  ( C# -- REG# )
    DUP 0= IF DROP 2C8 ELSE 2C3 + ENDIF ;
: LUMINACE+!  ( C# N -- )
    >R >CREG DUP C@ DUP R> + 0F AND
    SWAP F0 AND OR SWAP C!
;
: DARKER  -2 LUMINACE+! ;   ( C# -- )
: BRIGHTER 2 LUMINACE+! ;   ( C# -- )
: TXT-LOCATE  ROW-TXT C! COL-TXT ! ;   ( X Y -- )
: SCR-LOCATE  ROW-SCR C! COL-SCR ! ;   ( X Y -- )


10 LOAD

```

```
SCR #14
( ERROR MESSAGES     fig-FORTH )
Stack empty
Dictionary full
Wrong address mode
Isn't unique
Value error
Disk address error
Stack full
Disk Error!
All's well
```

```
SCR #15
( ERROR MESSAGES      fig-FORTH )
Use only in Definitions
Execution only
Conditionals not paired
Definition not finished
In protected dictionary
Use only when loading
Off current screen
Declare VOCABULARY
Nothing is wrong at all
```

```
SCR #16
( GRAPHICS -- SET-DEFAULTS )

: SET-DEFAULTS   ( -- )
    28 2C4 C!  CA 2C5 C! 94 2C6 C!
    46 2C7 C!  0 2C8 C! 0 0 OFFSET!
    0 ROW-SCR C!  0 COL-SCR C!
    0 ROW-TXT C!  0 COL-TXT C!
    1 USECOLOR 4 SCROLL-LINES C!
    0 TXT-MODE C! BF60 TXT-MEMORY !
    CURR-MODE C@ DUP PX/BY C@ PXBY C! PMODE C!
;

-->
```

SCR #17

```
( GRAPHICS -- MODE-LINE )
0 VARIABLE DL-HERE
: MODE-LINE  ( SM-ADDR MODE-NO MODE-LINES --  )
      OVER 40 + DL-HERE @ C!
      ROT DL-HERE @ 1+ !
      1 - SWAP OVER DL-HERE @
      3 + DUP ROT + DL-HERE !
      ROT ROT OVER IF FILL ELSE 2DROP DROP ENDIF
;
: BEGIN-DL  ( DL-ADDR -- DL-ADDR )
      DUP DUP 3 70 FILL 3 + DL-HERE !
;
: END-DL  ( DL-ADDR -- )
      DL-HERE @  41 OVER C! 1+ !
;
-->
```

SCR #18

```
( GRAPHICS -- MODE0 )

: MODE0  ( -- )
    BC20 BEGIN-DL
    BC40 2 18 MODE-LINE
    END-DL  2 CURR-MODE C!
    SET-DEFAULTS  18 SCROLL-LINES C!
    BC40 3C0 0 FILL 0 0 27 13 TWINDOW
    0 SCR-MODE C! BC40 SCR-MEMORY !
    BC40 TXT-MEMORY ! TW>W
    SCREEN-OFF BC20 SDLSTL ! SCREEN-ON
;
-->
```

```
SCR #19
( GRAPHICS -- MODE1 )
: MODE1   ( -- )
       BD5E BEGIN-DL
       BD80 6 14 MODE-LINE  BF60 2 4 MODE-LINE
       END-DL 6 CURR-MODE C!  SET-DEFAULTS
       1 SCR-MODE C! BD80 SCR-MEMORY !
       BD80 280 0 FILL 0 0 13 13 TWINDOW
       TW>W SCREEN-OFF BD5E SDLSTL ! SCREEN-ON
;
-->
```

```
SCR #20
( GRAPHICS -- MODE2 )
: MODE2   ( -- )
       BE58 BEGIN-DL
       BE70 7 A MODE-LINE  BF60 2 4 MODE-LINE
       END-DL 7 CURR-MODE C!  SET-DEFAULTS
       2 SCR-MODE C! BE70 SCR-MEMORY !
       BE70 190 0 FILL 0 0 13 9 TWINDOW TW>W
       SCREEN-OFF BE58 SDLSTL ! SCREEN-ON
;
-->
```

```
SCR #21
( GRAPHICS -- MODE3 )

: MODE3   ( -- )
        BE4E BEGIN-DL
        BE70 8 14 MODE-LINE  BF60 2 4 MODE-LINE
        END-DL 8 CURR-MODE C!  SET-DEFAULTS
        3 SCR-MODE C! BE70 SCR-MEMORY !
        BE70 190 0 FILL 0 0 27 13 TWINDOW TW>W
        SCREEN-OFF BE4E SDLSTL ! SCREEN-ON
;




-->
```

```
SCR #22
( GRAPHICS -- MODE4 )

: MODE4   ( -- )
        BD4A BEGIN-DL
        BD80 9 28 MODE-LINE  BF60 2 4 MODE-LINE
        END-DL 9 CURR-MODE C!  SET-DEFAULTS
        4 SCR-MODE C! BD80 SCR-MEMORY !
        BD80 280 0 FILL 0 0 4F 27 TWINDOW TW>W
        SCREEN-OFF BD4A 230 ! SCREEN-ON
;

-->
```

SCR #23

```
( GRAPHICS -- MODE5 )

: MODE5  ( -- )
      BB6A BEGIN-DL
      BBA0 A 28 MODE-LINE  BF60 2 4 MODE-LINE
      END-DL A CURR-MODE C!  SET-DEFAULTS
      5 SCR-MODE C! BBA0 SCR-MEMORY !
      BBA0 460 0 FILL 0 0 4F 27 TWINDOW TW>W
      SCREEN-OFF BB6A SDLSTL ! SCREEN-ON
;

-->
```

SCR #24

```
( GRAPHICS -- MODE6 )

: MODE6 ( -- )
      B782 BEGIN-DL
      B7E0 B 50 MODE-LINE  BF60 2 4 MODE-LINE
      END-DL B CURR-MODE C!  SET-DEFAULTS
      6 SCR-MODE C! B7E0 SCR-MEMORY !
      B7E0 820 0 FILL 0 0 9F 4F TWINDOW TW>W
      SCREEN-OFF B782 SDLSTL ! SCREEN-ON
;

-->
```

```
SCR #25
( GRAPHICS -- MODE7 )
: MODE7  ( -- )
      AFA2 BEGIN-DL
      B060 D 50 MODE-LINE  BF60 2 4 MODE-LINE
      END-DL D CURR-MODE C!  SET-DEFAULTS
      7 SCR-MODE C! B060 SCR-MEMORY !
      B060 FA0 0 FILL 0 0 9F 4F TWINDOW TW>W
      SCREEN-OFF AFA2 SDLSTL ! SCREEN-ON
;
-->
```

```
SCR #26
( GRAPHICS -- MODE8 )
: MODE8  ( -- )
      A050 BEGIN-DL
      A150 F 5D MODE-LINE B000 F 42 MODE-LINE
      BF60 2 4 MODE-LINE
      END-DL F CURR-MODE C! SET-DEFAULTS
      8 SCR-MODE C! A150 SCR-MEMORY !
      A150 1EB0 0 FILL 0 0 13F 9F TWINDOW
      TW>W SCREEN-OFF A050 SDLSTL ! SCREEN-ON
;
-->
```

```
SCR #27
( GRAPHICS -- LOCATE )
: LOCATE  OLDROW ! OLDCOL ! ;   ( X Y -- )
: READXY  OLDCOL @ OLDROW @ ;   ( -- X Y )
: RESORT   ( X1 Y1 X2 Y3 -- SLX SLY SRX SRY )
    ROT >R >R 2DUP > IF SWAP ENDIF R> R>
    2DUP > IF SWAP ENDIF >R SWAP R>
;
: POSITION    ( X Y -- ADDR RPX )
    CURR-MODE C@ BY/LN C@ *
    SCR-MEMORY @ + SWAP PXBY C@
    /MOD SWAP >R + R>
;
CODE SETCMSK   ( PX -- )
  PXBY LDA, CLC, 0 ,X ADC, TAY, DEY,
  PXMSKTB ,Y LDA, CMSK STA, POP JMP,
-->
```

```
SCR #28
( GRAPHICS -- PLOT )
CODE PXPUT  ( ADDR   --   )
  CMSK LDA, FF # EOR, 0 X) AND, 0 X) STA,
  CMSK LDA, COLORMSK AND, 0 X) ORA, 0 X) STA,
  OLDCHR STA, 0 ,X LDA, OLDADR STA,
  1 ,X LDA, OLDADR 1+ STA, POP JMP,
: WTCRS   ( -- )
    OLDCOL @ OLDROW @ +OFFSET 2DUP BOUNDARY?
   IF POSITION SETCMSK PXPUT ELSE 2DROP ENDIF
;
: PLOT  LOCATE WTCRS ;   ( X Y -- )
-->
```

SCR #29

```
( GRAPHICS -- DRAWLINE )
0 VARIABLE XSIGN  0 VARIABLE YSIGN
0 VARIABLE XYSIGN 0 VARIABLE VL
: HCALINC  ( -- INC1 INC2 DEFF XEND X  )
    TNEWCOL @ TOLDCOL @ - DUP >R TNEWROW @ TOLDROW @ -
    DUP ROT XYSIGN @ * - OVER OVER + XSIGN @ * ROT
    DUP + XSIGN @ * ROT DUP + XSIGN @ * ROT R> 0
;
: >TEMP  ( X1 Y1 X2 Y2 -- )
    VL @ IF TNEWCOL ! TNEWROW ! TOLDCOL ! TOLDROW !
                XSIGN @ YSIGN @ XSIGN ! YSIGN !
          ELSE TNEWROW ! TNEWCOL ! TOLDROW ! TOLDCOL !
          ENDIF XSIGN @ YSIGN @ +- XYSIGN !
;
-->
```

SCR #30

```
( GRAPHICS -- DRAWLINE )
: HDRAWPTS  ( INC1 INC2 DEFF XEND X -- )
    >R >R 0 R> R>
    DO I XSIGN @ + >R >R DUP XYSIGN @ +- 0<
      IF >R OVER R> + R>
      ELSE OVER + R> YSIGN @ +
      ENDIF R> OVER VL @ IF SWAP ENDIF PLOT
    XSIGN @ +LOOP 2DROP 2DROP
;
-->
```

SCR #31

```
( GRAPHICS -- SETMSK )
CODE SETMSK ( R2 R1 -- )
   PXBY LDA, CLC, 0 ,X ADC, TAY, DEY,
   XSIGN BIT, IFPL, RFMSKTB ,Y LDA, ENDIF,
   XSIGN BIT, IFMI, LFMSKTB ,Y LDA, ENDIF,
   FMSK STA, PXBY LDA, CLC, 2 ,X ADC, TAY,
   DEY, XSIGN BIT, IFPL, LFMSKTB ,Y LDA, ENDIF,
   XSIGN BIT, IFMI, RFMSKTB ,Y LDA, ENDIF,
   FMSK 1+ STA, POPTWO JMP,
CODE PX! ( ASM ONLY )
   TYA, PHA, 0 # LDY, CMSK LDA, FF # EOR,
   OLDADR )Y AND, OLDADR )Y STA, CMSK LDA,
   COLORMSK AND, OLDADR )Y ORA,
   OLDADR )Y STA, OLDCHR STA, PLA, TAY, RTS,

-->
```

SCR #32

```
( GRAPHICS -- INCADR VDRAW )
CODE INCADR ( ASM ONLY WITH JSR )
   OLDADR LDA, CLC, XSIGN ADC,
   OLDADR STA, OLDADR 1+ LDA,
   XSIGN 1+ ADC, OLDADR 1+ STA, RTS,

CODE VDRAW ( DISP +-B/L -- )
   2 ,X LDA, IFNE, TAY, BEGIN,
   OLDADR LDA, CLC, 0 ,X ADC,
   OLDADR STA, OLDADR 1+ LDA, 1 ,X ADC,
   OLDADR 1+ STA, ' PX! JSR, DEY,
   0= END, ENDIF, POPTWO JMP,



-->
```

```
SCR #33
( GRAPHICS -- HDRAW )
CODE HDRAW  ( ADDR1 DISP -- )
   2 ,X LDA, OLDADR STA, 3 ,X LDA,
   OLDADR 1+ STA, 0 ,X LDA, IFEQ,
   FMSK LDA, FMSK 1+ AND, CMSK STA,
   ' PX! JSR, ENDIF, 0 ,X LDA,
   IFNE, FMSK LDA, CMSK STA, ' PX! JSR,
   ' INCADR JSR, XSAVE STX, 0 ,X LDA,
   TAX, DEX, IFNE, BEGIN, COLORMSK LDA,
   OLDADR )Y STA, ' INCADR JSR,
   DEX, 0= END, ENDIF, FMSK 1+ LDA,
   CMSK STA, ' PX! JSR, XSAVE LDX,
   ENDIF, POPTWO JMP,

-->

SCR #34
( GRAPHICS -- DIRCT )
: OUTCODE  ( X Y -- CODE )
    0 OVER YMAX @ > IF 8 + ENDIF
    SWAP YMIN @ < IF 4 + ENDIF
    OVER XMAX @ > IF 2 + ENDIF
    SWAP XMIN @ < IF 1+ ENDIF
;
: DIR?   ( N1 N2 -- SIGN )
    - DUP 0=
    IF DROP 0
    ELSE 0< IF -1 ELSE 1 ENDIF
    ENDIF
;


-->
```

SCR #35

```
( GRAPHICS -- DIRCTION? )
: DIRECTION  ( X1 Y1 X2 Y2  -- XSIGN YSIGN  )
    ROT DIR? DUP YSIGN !
    >R SWAP DIR? DUP XSIGN ! R>
;
: XCLIP   ( X -- CX )
    XMIN @ MAX XMAX @ MIN
;
: YCLIP   ( Y -- CY )
    YMIN @ MAX XMAX @ MIN
;
: HCLIP   ( X1 Y1 X2 Y2  -- CX1 CY1 CX2 CY2 )
    >R XCLIP >R >R XCLIP R> R> R>
;

-->
```

SCR #36

```
( GRAPHICS -- HLINE )
: VCLIP   ( X1 Y1 X2 Y2 -- CX1 CY1 CX2 CY2 )
    YCLIP >R >R YCLIP R> R>
;
: HLINE  ( X1 Y1 X2 Y2  --  )
    HCLIP >R >R POSITION
    R> R> ROT >R POSITION
    >R OVER - ABS R> R> SETMSK HDRAW
;
: VLINE  ( X1 Y1 X2 Y2  --  )
    VCLIP SWAP DROP OVER - ABS >R
    POSITION SETCMSK PXPUT R> CURR-MODE @
    BY/LN C@ YSIGN @ +- VDRAW
;

-->
```

SCR #37

```
( GRAPHICS -- CLINE )
: CLINE ( X1 Y1 X2 Y2 -- )
   2OVER 2OVER 2OVER OFFSET>T OFFSET! ROT - ABS
   >R - ABS R> < VL !
   >TEMP HCALINC 0 0 PLOT HDRAWPTS T>OFFSET
;
: DRAWLINE ( -- )
   OLDCOL @ OLDROW @ +OFFSET 2DUP OUTCODE >R
   NEWCOL @ NEWROW @ +OFFSET 2DUP OUTCODE R> AND 0=
   IF 2OVER 2OVER DIRECTION 0=
      IF DROP HLINE
      ELSE 0= IF VLINE ELSE CLINE ENDIF
      ENDIF
   ELSE 2DROP 2DROP
   ENDIF NEWCOL @ OLDCOL ! NEWROW @ OLDROW !
; -->
```

SCR #38

```
( GRAPHIC -- LINE , DRAWTO )
: LINE ( X1 Y1 X2 Y2 -- )
   NEWROW ! NEWCOL ! OLDROW ! OLDCOL ! DRAWLINE
;
: DRAWTO ( X2 Y2 -- )
   NEWROW ! NEWCOL ! DRAWLINE
;

0 VARIABLE FILLFLG
: CIRCLE-FILL ( X Y -- )
   2DUP OVER MINUS OVER LINE SWAP
   2DUP OVER MINUS OVER LINE MINUS
   2DUP OVER MINUS OVER LINE
   SWAP MINUS OVER MINUS OVER LINE
;
-->
```

```
SCR #39
 ( GRAPHICS -- CIRCLE )

: CIRCLE-PTS ( X Y -- )
    OVER OVER PLOT SWAP OVER OVER
    PLOT MINUS OVER OVER PLOT
    OVER OVER MINUS SWAP MINUS
    OVER OVER PLOT OVER OVER SWAP
    MINUS OVER OVER SWAP PLOT PLOT
    SWAP PLOT SWAP PLOT
;
: CIRCLE-PUT ( X Y -- )
   FILLFLG C@ IF CIRCLE-FILL
              ELSE CIRCLE-PTS
              ENDIF
;
-->
```

```
SCR #40
 ( GRAPHICS -- CIRCLE )
: DRAWCIRCLE ( X Y R -- )
    >R OFFSET>T +OFFSET OFFSET! R>
    3 OVER 2 * - SWAP 0 SWAP
    BEGIN OVER OVER <
    WHILE OVER OVER CIRCLE-PUT OVER OVER
      >R >R >R >R DUP 0<
      IF 4 R> * 6 + + R> DROP R> 1+ R>
      ELSE 4 R> R> - * + 10 + R> 1+ R> 1 -
    ENDIF
    REPEAT OVER OVER =
    IF CIRCLE-PUT DROP
    ELSE DROP DROP DROP
    ENDIF T>OFFSET
;
-->
```

SCR #41

```
( GRAPHICS -- CIRCLE )

: CIRCLE   ( XC YC R -- )
    >R 2DUP R> 0 FILLFLG C!
    DRAWCIRCLE OLDROW ! OLDCOL !
;
: FCIRCLE  ( XC YC R -- )
    >R 2DUP R> 1 FILLFLG C!
    DRAWCIRCLE OLDROW ! OLDCOL !
;
: BOX  ( X1 Y1 X2 Y2 -- )
    NEWROW ! NEWCOL ! 2DUP OLDROW ! OLDCOL !
    NEWCOL @ 2DUP NEWROW @ OLDCOL @ 2DUP
    OLDROW @ LOCATE DRAWTO DRAWTO DRAWTO DRAWTO
;
-->
```

SCR #42

```
( GRAPHICS -- FBOX )
: HPOS  ( X1 Y1 X2 Y2 -- ADDR1 DX )
    >R >R POSITION OVER R> R> POSITION
    >R SWAP - SWAP R> SWAP SETMSK
;
: FILBOX  ( -- )
    OLDCOL @ OLDROW @ +OFFSET 2DUP OUTCODE >R
    NEWCOL @ NEWROW @ +OFFSET 2DUP OUTCODE R> AND 0=
    IF RESORT 2OVER 2OVER DIRECTION 2DROP YCLIP >R
      XCLIP >R YCLIP >R XCLIP R> R> OVER R> OVER - >R
      HPOS CURR-MODE C@ BY/LN C@ R> OVER OVER *
      >R +- DUP R> + >R SWAP ROT DUP R> + SWAP
      DO I OVER HDRAW OVER +LOOP 2DROP
    ENDIF
;
-->
```

```
SCR #43
( GRAPHICS -- CLRBOX )
: FBOX ( X1 Y1 X2 Y2 -- )
    NEWROW ! NEWCOL ! OLDROW ! OLDCOL !
    FILBOX NEWCOL @ OLDCOL ! NEWROW @ OLDROW !
;
: BCLEAR ( X1 Y1 X2 Y2 -- )
    CURR-COLOR C@ >R 0 USECOLOR
    FBOX R> USECOLOR
;
: WCLEAR ( -- )
    XMIN @ YMIN @ XMAX @ YMAX @ BCLEAR
;
: CLEAR ( -- )
    TXMIN @ TYMIN @ TXMAX @ TYMAX @ BCLEAR
;
-->
```

```
SCR #44
( GRAPHICS -- BFILL )

0 VARIABLE BYLN   0 VARIABLE OLDCOLMSK
0 VARIABLE UPTRIG 0 VARIABLE DOWNTRIG
D2 CONSTANT STACK-PT 90 CONSTANT XTEMP
0 VARIABLE STACK-BTM
: READPX ( X Y -- COLOR )
    POSITION SETCMSK C@ CMSK C@ AND 0
    BEGIN 2DUP GETCOLMSK CMSK C@ AND = NOT
    WHILE 1+
    REPEAT SWAP DROP
;
: SETBYLN ( -- )
    CURR-MODE C@ BY/LN C@ BYLN C!
;
-->
```

```
SCR #45
( GRAPHICS -- BFILL )

CODE (=OLDCOLOR?)   ( ADDR -- F ) ( ASM ONLY )
    0 # LDY, 0 X) LDA, CMSK AND, 0 ,X STA,
    OLDCOLMSK LDA, CMSK AND, 0 ,X CMP,
    1 ,X STY, IFEQ, INY, ENDIF,
    0 ,X STY, RTS,

CODE =OLDCOLOR?  ( ADDR -- F )
  ' (=OLDCOLOR?) JSR, NEXT JMP,

: SETSTACK  ( ADDR -- )
    DUP STACK-BTM ! STACK-PT !
;
-->
```

```
SCR #46
( GRAPHICS -- BFILL )
: STACK-EMPTY?   ( -- F )
    STACK-BTM @ STACK-PT @ < NOT
;
: STACK-POP      ( -- X Y )
    STACK-PT @ 2 - DUP @ SWAP
    2 - DUP STACK-PT ! @
;
CODE (STACK-PUSH)     ( X Y -- ) ( ASM ONLY )
  XTEMP STX, 0 # LDA, XTEMP 1+ STA, 4 # LDY,
  BEGIN, DEY, XTEMP )Y LDA, STACK-PT )Y STA, TYA,
  0= UNTIL, STACK-PT LDA, CLC, 4 # ADC,
  STACK-PT STA, STACK-PT 1+ LDA,
  0 # ADC, STACK-PT 1+ STA,
  INX, INX, INX, INX, RTS,
-->
```

```
SCR #47
( GRAPHICS -- BFILL )
CODE STACK-PUSH   ( X Y -- )
  ' (STACK-PUSH) JSR, NEXT JMP,
CODE ABOVE?  ( ADDR X Y -- )
  0 ,X LDA, SEC, 1 # SBC, 0 ,X STA, 1 ,X LDA, 0 # SBC,
  0 # SBC, 1 ,X STA, SEC, 0 ,X LDA, YMIN SBC,
  1 ,X LDA, YMIN 1+ SBC, IFVS, 80 # EOR, ENDIF,
  IFMI, INX, INX, POPTWO JMP, ENDIF,
  XSAVE STX, INX, INX, INX, INX, 0 ,X LDA,
  SEC, BYLN SBC, 0 ,X STA, 1 ,X LDA, 0 # SBC,
  1 ,X STA, ' (=OLDCOLOR?) JSR, 0 ,X LDA,
  IFEQ, 1 # LDA, UPTRIG STA, POP JMP, ENDIF,
  UPTRIG LDA, IFEQ, POP JMP, ENDIF,
  0 # LDA, UPTRIG STA, XSAVE LDX,
  ' (STACK-PUSH) JSR, POP JMP,
-->
```

```
SCR #48
( GRAPHICS -- BFILL )

CODE BELOW?  ( ADDR X Y -- )
  0 ,X LDA, CLC, 1 # ADC, 0 ,X STA, 1 ,X LDA,
  0 # ADC, 1 ,X STA, SEC, YMAX LDA, 0 ,X SBC,
 YMAX 1+ LDA, 1 ,X SBC, IFVS, 80 # EOR, ENDIF,
  IFMI, INX, INX, POPTWO JMP, ENDIF,
  XSAVE STX, INX, INX, INX, INX, 0 ,X LDA,
  CLC, BYLN ADC, 0 ,X STA, 1 ,X LDA, 0 # ADC,
  1 ,X STA, ' (=OLDCOLOR?) JSR, 0 ,X LDA,
  IFEQ, 1 # LDA, DOWNTRIG STA, POP JMP, ENDIF,
  DOWNTRIG LDA, IFEQ, POP JMP, ENDIF,
  0 # LDA, DOWNTRIG STA, XSAVE LDX,
  ' (STACK-PUSH) JSR, POP JMP,

-->
```

```
SCR #49
( GRAPHICS  --  BFILL  )
: <FILL   ( X Y -- ADR PXR BX BY )
    2DUP >R >R POSITION
    BEGIN 2DUP SETCMSK =OLDCOLOR?
      I XMIN @ < NOT AND
    WHILE R> 1 - >R DUP 0=
      IF DROP 1 - PXBY C@ ENDIF 1 -
    REPEAT R> R> 1 DUP UPTRIG C!
      DOWNTRIG C!
;
-->
```

```
SCR #50
(  GRAPHICS  --  BFILL  )
: FILL>   ( ADDR PXR X Y --  )
    >R >R
    BEGIN 1+ DUP PXBY C@ =
      IF DROP 1+ 0 ENDIF
      2DUP SETCMSK =OLDCOLOR? R> 1+
      DUP >R XMAX @ > NOT AND
    WHILE OVER PXPUT OVER 2R ABOVE?
      OVER 2R BELOW?
    REPEAT 2DROP R> R> 2DROP
;
-->
```

SCR #51

```
( GRAPHICS -- BFILL )
: BFILL    ( X Y -- )
    SETBYLN 2DUP READPX GETCOLMSK
    DUP OLDCOLMSK C! COLORMSK C@ = NOT
    IF HERE 100 + SETSTACK STACK-PUSH
      BEGIN STACK-EMPTY? NOT
      WHILE STACK-POP
            <FILL
            FILL>
      REPEAT
    ELSE 2DROP
    ENDIF
;
-->
```

SCR #52

```
( LSHIFT )
0 VARIABLE SADDR    0 VARIABLE DADDR
0 VARIABLE DBYTES   0 VARIABLE BITS
0 VARIABLE DLPX     0 VARIABLE DRPX
0 VARIABLE SDXA
400 VARIABLE WORKING 92 CONSTANT WADDR

CODE LSHIFT  ( ADDR BYTES BITS -- )
  4 ,X LDA, WADDR STA, 5 ,X LDA, WADDR 1+ STA,
  BEGIN, 2 ,X LDY, CLC,
    BEGIN, DEY, WADDR )Y LDA, ROL.A,
      WADDR )Y STA, TYA, 0=
    UNTIL, 0 ,X DEC, 0=
  UNTIL,
  INX, INX, POPTWO JMP,
-->
```

```
SCR #53
( RSHIFT )
CODE RSHIFT  ( ADDR BYTES BITS  --  )
  4 ,X LDA, WADDR STA, 5 ,X LDA,
  WADDR 1+ STA,
  BEGIN, 0 # LDY, 2 ,X LDA,
    XSAVE STX, TAX, CLC,
    BEGIN, WADDR )Y LDA, ROR.A,
      WADDR )Y STA, INY, DEX, 0=
    UNTIL,
    XSAVE LDX, 0 ,X DEC, 0=
  UNTIL,
  INX, INX, POPTWO JMP,
-->
```

```
SCR #54
( RCOPY  -- )
CODE RMOST ( bfr raddr --  )
  PXBY LDA, CLC, DRPX ADC,
  TAY, DEY, LFMSKTB ,Y LDA, 2 X) AND,
  2 X) STA, LFMSKTB ,Y LDA, FF # EOR,
  0 X) AND, 2 X) ORA, 2 X) STA,
  POPTWO JMP,
CODE LMOST  ( bfl laddr  --  )
  PXBY LDA, CLC, DLPX ADC,
  TAY, DEY, RFMSKTB ,Y LDA,
  2 X) AND, 2 X) STA, RFMSKTB ,Y LDA,
  FF # EOR, 0 X) AND, 2 X) ORA,
  2 X) STA, POPTWO JMP,
FORTH
-->
```

```
SCR #55
( RCOPY -- )

: LMOVE ( -- )
    SADDR @ WORKING @ DBYTES C@ 1+ CMOVE
    WORKING @ DBYTES C@ 1+ BITS @ DUP
    IF DUP 0<
       IF MINUS LSHIFT
       ELSE RSHIFT
       ENDIF
    ELSE DROP 2DROP
    ENDIF WORKING @ DADDR @ 2DUP LMOST
    DBYTES C@ >R OVER R + 1 - OVER
    R + 1 - RMOST R> CMOVE
;
-->

SCR #56
( RCOPY -- )

: PREPARE ( slx sly srx sry dlx dly -- dyend dly )
    >R >R RESORT 2OVER 2OVER DIRECTION 2DROP
    >R OVER R> SWAP - >R >R 2DUP OVER R> SWAP
    OVER SWAP - >R OVER HPOS SDXA ! SADDR !
    POSITION SWAP DROP R> R> SWAP R + TXMAX @
    MIN R> R POSITION 2DUP DLPX C!
    DADDR ! ROT R POSITION DRPX C!
    SWAP >R SWAP - 1+ DBYTES C! R>
    SWAP >R SWAP - CURR-MODE C@ BI/PX C@ *
    BITS ! R> R + TYMAX @ MIN 1+ R>
;


-->
```

SCR #57

```
( RCOPY )
: RCOPY ( slx sly srx sry dlx dly -- )
    PREPARE
    DO LMOVE CURR-MODE C@ BY/LN C@
       DUP SADDR +! DADDR +!
    LOOP
;
: RMOVE ( slx sly srx sry dlx dly -- )
    CURR-COLOR C@ >R 0 USECOLOR PREPARE
    DO LMOVE SADDR @ SDXA @ HDRAW
       CURR-MODE C@ BY/LN C@ DUP
       SADDR +! DADDR +!
    LOOP R> USECOLOR
;
-->
```

SCR #58

```
(    TEXT    ----    3/16/85    )
E000 CONSTANT CHBASE
PXMSKTB 7 + CONSTANT BITMSKTB
: >INCODE     ( ascii -- incode )
    DUP 20 < DUP 7F > OR
    IF DROP 0
    ELSE DUP 60 <
      IF 20 - ENDIF
    ENDIF
;
-->
```

SCR #59

```
(     TEXT    ----     )
: CHARACTER   ( X Y CODE -- )
    >INCODE 8 * CHBASE + SWAP 1+ DUP 8 -
    DO SWAP 8 O
      DO OVER C@ I BITMSKTB + C@ AND
        IF DUP I + J PLOT ENDIF
      LOOP SWAP 1+
    LOOP 2DROP
;
: PRINT     ( X Y ADDR LENGTH -- )
    OVER + SWAP
    DO 2DUP I C@ CHARACTER
       SWAP 8 + SWAP
    LOOP 2DROP
;
-->
```

SCR #60

```
(     TEXT    ----     )
: (PRINT)     ( X Y -- )
    R COUNT DUP 1+ R> + >R PRINT
;
: STRING"     ( X Y -- )
    22 STATE @
    IF COMPILE (PRINT) WORD HERE C@ 1+ ALLOT
    ELSE WORD HERE COUNT PRINT
    ENDIF
; IMMEDIATE
: TEXT    ( X Y -- )
    CR ." TYPE IN TEXT STRING : " CR
    QUERY O WORD HERE COUNT PRINT
;

-->
```

```
SCR #61
( GRAPHICS -- DLI )
200 CONSTANT VDSLST D40E CONSTANT NMIEN
E45C CONSTANT SETVBV C28A CONSTANT VBDEXIT
: DLI-ON  80 NMIEN C@ OR NMIEN C! ;  ( -- )
: DLI-OFF 7F NMIEN C@ AND NMIEN C! ;  ( -- )
: SETDLIV  VDSLST ! ;  ( DLIRTN -- )
CODE SETVBD  ( ADDR -- )
  XSAVE STX, 0 ,X LDY, 1 ,X LDA,
  TAX, 7 # LDA, SETVBV JSR,
  XSAVE LDX, POP JMP,
CODE RESETVBD  (  --  )
  XSAVE STX, 7 # LDA, C2 # LDX,
  8A # LDY, SETVBV JSR,
  XSAVE LDX, NEXT JMP,
-->

SCR #62
( GRAPHICS -- GSAVE )
0 VARIABLE PDLA   0 VARIABLE CDLA
0 VARIABLE PDDA   0 VARIABLE CDDA
1 VARIABLE DLFLAG 1 VARIABLE DDFLAG
400 RECORD MFD
    % TOT#SEC   2 BYTES  % USED#SEC  2 BYTES
    % DIR-SEC   2 BYTES  % DIR#SEC   2 BYTES
    % FREE-LIST 2 BYTES  % FREE#SEC  2 BYTES
    END-RECORD
580 RECORD FDS
    % SEC#  2 BYTES   % NEXT   2 BYTES
    % LDA   2 BYTES   % BTS   2 BYTES
    % DATA  120 BYTES
    END-RECORD
-->
```

```
SCR #63
( GRAPHICS -- GSAVE )

500 VARIABLE FDA
502 RECORD FDD
      % LENGTH  1 BYTES  % NAME    5 BYTES
      % DLA     2 BYTES  % HEAD    2 BYTES
      % TAIL    2 BYTES  % #SEC    1 BYTES
      % SCROLL  1 BYTES  % SCRM    2 BYTES
      % TXTM    2 BYTES  % FXMIN   2 BYTES
      % FXMAX   2 BYTES  % FYMIN   2 BYTES
      % FYMAX   2 BYTES  % COLR    5 BYTES
      END-RECORD


-->

```

```
SCR #64
(  GSAVE   --   )

: READ   1 R/W ;   ( ADDR SEC# -- )
: WRITE 0 R/W ;   ( ADDR SEC# -- )
: GET-MFD MFD @ 0 READ ; ( -- )
: PUT-MFD MFD @ 0 WRITE ; ( -- )
: PUT-FDD FDA @ DUP @ WRITE ; ( -- )
: PUT-FDS FDS @ SEC# @ WRITE ; ( -- )
: FORMAT-MFD ( -- )
    MFD @ 80 0 FILL 2CE TOT#SEC !
    0 USED#SEC ! 1 DIR-SEC ! F DIR#SEC !
    10 FREE-LIST ! 2BE FREE#SEC !
    PUT-MFD
;
-->

```

```
SCR #65
( GSAVE -- )
: FORMAT-FDD  ( -- )
   FDA @ 80 0 FILL 10 1
   DO I FDA @ ! FDA @ I WRITE LOOP
;
: FORMAT-FDS ( -- )
   FDS @ DUP 80 0 FILL 2CE 10
   DO DUP I DUP DUP SEC# ! 1+ NEXT ! WRITE LOOP
   2CE DUP SEC# ! 0 NEXT ! WRITE
;
: FORMAT  ( -- )
   CR ." Insert blank disk"
   CR ." Type 'Y' to start" CR KEY 59 =
   IF ." <WAIT>" CR FORMAT-MFD FORMAT-FDD
     FORMAT-FDS ENDIF
;  -->

SCR #66
( GSAVE -- )
: MATCH?  ( ADDR1 ADDR2 -- F )
   0 ROT ROT OVER C@ 1+
   6 MIN OVER + SWAP
   DO DUP C@ I C@ = NOT
     IF LEAVE SWAP DROP 1 SWAP ENDIF 1+
   LOOP DROP NOT
;
: RELEASE  (  --  )
   FDS @ TAIL @ 2DUP READ
   FREE-LIST @ NEXT ! WRITE #SEC C@
   DUP MINUS USED#SEC +! FREE#SEC +!
   HEAD @ FREE-LIST ! LENGTH 1F 0 FILL
;
-->
```

SCR #67

```
( GSAVE -- )
: FIND-FD  ( ADDR -- ADDR 1 or 0 )
    0 10 1
    DO DROP FDA @ DUP I READ 2 + 0 4 0
      DO DROP 2DUP MATCH?
          IF SWAP DROP 1 LEAVE
          ELSE 1F + 0
          ENDIF
      LOOP
      IF 1 LEAVE ELSE DROP 0 ENDIF
    LOOP
    IF 1 ELSE DROP 0 ENDIF
;
: FIND-FN  ( -- ADDR F )
    GET-MFD 0 WORD HERE FIND-FD ;
-->
```

SCR #68

```
( GSAVE  -- )
: FIND-EMPTY  (  -- ADDR 1 Or 0 )
    0 10 1
    DO DROP FDA @ DUP I READ 2 + 0 4 0
      DO DROP DUP C@ 0=
        IF 1 LEAVE ELSE 1F + 0 ENDIF
      LOOP
      IF 1 LEAVE ELSE DROP 0 ENDIF
    LOOP
;
: (DELETE)  ( ADDR F -- )
    IF FDD ! RELEASE PUT-FDD PUT-MFD
    ELSE ." File name Not Found ! "
    ENDIF
;
-->
```

SCR #69

```
( GSAVE  --  )
: DELETE  FIND-FN (DELETE) ;   ( -- )

: REQUEST  (  --  )
    FREE#SEC @ 0=
    IF ." Disk Full ! " RELEASE
      PUT-FDD PUT-MFD ABORT
    ENDIF
    1 USED#SEC +! -1 FREE#SEC +!
    FREE-LIST @ DUP HEAD @
    IF NEXT ! PUT-FDS ELSE HEAD ! ENDIF
    TAIL ! FDS @ FREE-LIST @ READ
    NEXT @ FREE-LIST ! 0 NEXT !
    1 #SEC +!
;
-->
```

SCR #70

```
( GSAVE  --  )
: STORE-DL  (  --  )
    DLFLAG C@ IF PDLA @ CDLA @ OVER - DUP
    IF REQUEST 78 MIN >R DUP LDA !
      DUP DATA R CMOVE R + PDLA ! R> BTS !
    ELSE 2DROP
    ENDIF ENDIF
;
: STORE-DD  (  --  )
    DDFLAG C@ IF PDDA @ CDDA @ OVER - DUP
    IF REQUEST 78 MIN >R DUP LDA !
      DUP DATA R CMOVE R + PDDA ! R> BTS !
    ELSE 2DROP
    ENDIF ENDIF
;
-->
```

SCR #71

```
( GSAVE -- )
: STORE-DATA  ( ADDR -- )
    DUP CDLA ! PDLA !
    BEGIN 0 CDLA @ C@ DUP 0F AND DUP
      IF OVER 40 AND
        IF STORE-DD DUP 1 >
          IF CDLA @ 1+ @ DUP PDDA ! CDDA !
            3 CDLA +! BY/LN C@ CDDA +! DROP
          ELSE 2DROP CDLA DUP @ 1+ @ SWAP
            2DUP @ 3 CDLA +! STORE-DL >
            IF ! ELSE 2DROP DROP 1 ENDIF
          ENDIF
        ELSE 1 CDLA +! BY/LN C@ CDDA +! DROP
        ENDIF
      ELSE 2DROP 1 CDLA +!
      ENDIF   -->
```

SCR #72

```
( GSAVE -- )
      CDDA @ PDDA @ - 77 >
      IF STORE-DD ENDIF
      CDLA @ PDLA @ - 77 >
      IF STORE-DL ENDIF
    UNTIL
    CDDA @ PDDA @ - 0 >
    IF STORE-DD ENDIF
    CDLA @ PDLA @ - 0 >
    IF STORE-DL ENDIF
    PUT-FDS
;


-->
```

SCR #73

```
( GSAVE -- )
: STORE-ENV ( -- )
    SCROLL-LINES C@ SCROLL C!
    SCR-MEMORY @ SCRM ! TXT-MEMORY @ TXTM !
    TXMIN @ FXMIN ! TXMAX @ FXMAX ! TYMIN @ FYMIN !
    TYMAX @ FYMAX ! 2C4 COLR 5 CMOVE
;
: LOAD-ENV ( -- )
    SCROLL C@ SCROLL-LINES C!
    SCRM @ SCR-MEMORY ! TXTM @ TXT-MEMORY !
    FXMIN @ TXMIN ! FXMAX @ TXMAX ! FYMIN @ TYMIN !
    FYMAX @ TYMAX ! COLR 2C4 5 CMOVE
;
-->
```

SCR #74

```
( GSAVE -- )
: (GSAVE) ( ADDR F -- )
    IF DROP ." File name Not Unique ! "
    ELSE HERE FIND-EMPTY
      IF FDD ! DUP C@ 1+ 8 MIN LENGTH
        SWAP CMOVE STORE-ENV SDLSTL @ DUP DLA !
        STORE-DATA PUT-FDD PUT-MFD
      ELSE ." Directory Full ! "
      ENDIF
    ENDIF
;

: GSAVE  FIND-FN (GSAVE) ;  ( -- )

-->
```

```
SCR #75
( GRAPHICS -- GSAVEALL )

: SAVE-DL  ( -- )
    1 DLFLAG C! 0 DDFLAG C! GSAVE
;
: SAVE-DD  ( -- )
    0 DLFLAG C! 1 DDFLAG C! GSAVE
;
: SAVE-ALL  ( -- )
    1 DLFLAG C! 1 DDFLAG C! GSAVE
;




-->
```

```
SCR #76
(  GLOAD  --  )
: LOAD-DATA  ( -- )
    FDS @ HEAD @
    BEGIN DUP
    WHILE OVER SWAP READ DATA
      LDA @ BTS @ CMOVE NEXT @
    REPEAT 2DROP
;
: (GLOAD)  ( -- )
    IF FDD ! LOAD-ENV LOAD-DATA DLA @
      SCREEN-OFF SDLSTL ! SCREEN-ON
    ELSE ." File name Not Found ! "
    ENDIF
;
: GLOAD  FIND-FN (GLOAD) ;  ( -- )
-->
```

```
SCR #77
( GRAPHICS -- DIR )
: DIR  ( -- )
    CR FDA @ 0 10 1
    DO OVER DUP I READ 2 + 4 0
       DO DUP C@
          IF SWAP DUP 3 =
             IF DROP 0 CR ELSE SPACE ENDIF
             1+ SWAP DUP COUNT DUP >R
             TYPE 5 R> - SPACES
          ENDIF 1F +
       LOOP DROP
    LOOP 2DROP
;


-->
```

```
SCR #78
(  PLAYER  )
HEX
D000 CONSTANT HPOSPO D008 CONSTANT SIZEPO
D01D CONSTANT GRACTL D407 CONSTANT RPMBS
3 CARRAY VPOSPO   3 CARRAY HLOWPO
3 CARRAY RPXPO    3 CARRAY RPYPO
3 ARRAY  FIGPO    3 ARRAY  ADDRPO
3 CARRAY HPSIZE
0 VARIABLE PVLOW  0 VARIABLE PHLOW
0 VARIABLE VLOWPO 0 VARIABLE CURR-PLAYER
0 VARIABLE CURR-PFIG 0 VARIABLE PADDR
8800 VARIABLE PMBASE 0 VARIABLE PSIZE
0 VARIABLE SLML 0 VARIABLE PXML
0 VARIABLE PNL
-->

```

SCR #79

```
( GRAPHICS -- PADXY )
270 CONSTANT PADX  271 CONSTANT PADY
27C CONSTANT LBTN  27D CONSTANT RBTN
0 VARIABLE PFX  0 VARIABLE PFY
CODE PADXY   ( -- X Y )
  DEX, DEX, DEX, DEX, 0 # LDA,
  1 ,X STA, 3 ,X STA, PADX LDA,
  2 ,X STA, PADY LDA, 0 ,X STA,
  BEGIN, 0 # LDY, PADX LDA, 2 ,X CMP,
    IFNE, INY, 2 ,X STA, ENDIF,
    PADY LDA, 0 ,X CMP,
    IFNE, INY, 0 ,X STA, ENDIF,
    0 # CPY, 0=
  UNTIL,  NEXT JMP,

-->
```

SCR #80

```
( PAD -- TEST )
: SETPF  ( -- )
    CURR-MODE C@ DUP PX/ML @ PFX !
    SL/ML C@ A0 SWAP / PFY !
;
: >SCRXY  ( PX PY -- X Y )
    PVLOW C@ - DUP 0 > IF PFY @ * A0 / ENDIF SWAP
    PHLOW C@ - DUP 0 >
      IF CURR-MODE C@ F =
        IF 2 * ELSE PFX @ * A0 / ENDIF
      ENDIF SWAP
;
0 VARIABLE PDATA1 -2 ALLOT
  0E C, 1010 , 1010 , 3838 , EEEE ,
  3838 , 1010 , 1010 ,
-->
```

SCR #81

```
( PLAYER )
0 VARIABLE PDATA0 -2 ALLOT
  07 C, 10 C, 10 C, 38 C, EE C, 38 C, 10 C, 10 C,
: SETPSIZE   ( P# CODE -- )
    2DUP 1+ SWAP HPSIZE C! OVER
    SIZEP0 + C! 30 SWAP HLOWP0 C!
;
: SINGLE-SIZE  0 SETPSIZE ;   ( P# -- )
: DOUBLE-SIZE  1 SETPSIZE ;   ( P# -- )
: QUAD-SIZE    3 SETPSIZE ;   ( P# -- )
: SETPLINE  ( CODE LINES LEN ADDR -- )
    OVER PSIZE ! 4 0
    DO DUP PMBASE @ + I ADDRP0 ! OVER + LOOP
    2DROP PNL C! DUP 10 + VLOWP0 C!
    EF SDMCTL C@ AND OR SDMCTL C!
;  -->
```

SCR #82

```
(  GRAPHICS -- PLAYER )
: SINGLE-LINE 10 1 100 400 SETPLINE ;   ( -- )
: DOUBLE-LINE 0 2 80 200 SETPLINE ;   ( -- )
: PLAYER-ON  (  -- )
    SDMCTL C@ 8 OR SDMCTL C!
    GRACTL C@ 2 OR GRACTL C!
;
: PLAYER-OFF  ( -- )
    SDMCTL C@ FB AND SDMCTL C!
    GRACTL C@ FD AND GRACTL C!
;
: PLAYER-CLEAR  ( -- )
    PADDR @ PSIZE @ 0 FILL ;
-->
```

SCR #83

```
( PLAYER )
: SETPFIG ( PDATA RPX RPY P# -- )
    >R I RPYPO C! I RPXPO C! R> FIGPO !
;
: SETPLAYERS ( -- )
    PMBASE @ DUP 100 / RPMBS C!
    800 0 FILL HPOSPO 4 0 FILL
    PCOLRO 4 FF FILL
    4 0 DO PDATAO 3 3 I SETPFIG I SINGLE-SIZE
        LOOP SINGLE-LINE PMODE C@ DUP
    SL/ML C@ SLML ! PX/ML @ PXML ! SETPF
;
-->
```

SCR #84

```
( PLAYER )
: USEPLAYER ( P# -- )
    DUP CURR-PLAYER C! DUP HLOWPO C@ OVER
    DUP RPXPO C@ SWAP HPSIZE C@ * - PHLOW C!
    VLOWPO C@ OVER RPYPO C@ - PVLOW C!
    DUP ADDRPO @ PADDR ! FIGPO @ CURR-PFIG !
;
: >PXY ( X Y -- PX PY )
    DUP 0 > IF SLML @ * PNL C@ / ENDIF PVLOW C@ +
    SWAP DUP 0 > IF A0 * PXML @ / ENDIF
    PHLOW C@ + SWAP
;
-->
```

SCR #85

```
( PLAYER )
: PLAYER ( X Y -- )
    >PXY PSIZE @ 2DUP <
    IF OVER - >R SWAP OVER CURR-PLAYER C@
      SWAP OVER VPOSPO C! HPOSPO + C!
      PLAYER-CLEAR PADDR @ +
      CURR-PFIG @ DUP
      >R 1+ SWAP R> C@ R> MIN CMOVE
    ELSE 2DROP DROP
    ENDIF
;
-->
```

SCR #86

```
( ZOOM )
0 VARIABLE ZMODE 0 VARIABLE ZFLAG
0 VARIABLE ZMLX  0 VARIABLE ZMLY
0 VARIABLE ZXLOW 0 VARIABLE ZYLOW
0 VARIABLE ZXHIG 0 VARIABLE ZYHIG
0 VARIABLE CK/PX 0 VARIABLE ZDL-ADDR
0 VARIABLE ZX0   0 VARIABLE ZY0
0 VARIABLE FIXFLAG

D404 CONSTANT HSCROL

-->
```

```
SCR #87
( ZOOM )
: !ZOOM-DLS  ( DLA SCRA MZ MB LINES -- )
    >R BY/LN C@ >R 10 + >R SWAP BEGIN-DL SWAP R> R>
    R> 0 DO >R 2DUP 1 MODE-LINE SWAP R + SWAP R>
    LOOP 2DROP DROP BF60 2 4 MODE-LINE END-DL
;

: FLUSHDL  ( ADDR PX -- )
    CK/PX C@ 4 = OVER 0 > AND NOT IF >R 1 - R> ENDIF
    4 SWAP - CK/PX C@ * HSCROL C!
    ZDL-ADDR @ SWAP OVER 4 +
    DUP ZMLY @ 3 * + SWAP
    DO DUP I ! CURR-MODE C@ BY/LN C@ + 3 +LOOP
    DROP SDLSTL SCREEN-OFF ! SCREEN-ON
;
-->
```

```
SCR #88
(  ZOOM  )
: >ZOOMXY  ( X Y -- ZX ZY )
    2DUP BOUNDARY?
    IF FIXFLAG C@
      IF ZXO @ ZYO @
      ELSE 2DUP ZMLY @ 2 / - ZYLOW @ MAX ZYHIG @ MIN
        SWAP ZMLX @ 2 / - ZXLOW @ MAX ZXHIG @ MIN SWAP
      ENDIF 2DUP POSITION FLUSHDL 2DUP ZYO ! ZXO !
      >R SWAP >R - R> R> -
    ENDIF
;
: ZPLAYER  ( X Y -- )
    ZFLAG C@ IF >ZOOMXY ENDIF PLAYER
;
FORTH DEFINITIONS DECIMAL
;S
```

APPENDIX C: GLOSSARY OF WORDS FOR FIGE

word screen number high/low level

stack effects

description

Comments

X will always represent the X-coordinate.

Y will always represent the Y-coordinate.

1. Constants for FIGE

COLPFO 7

(-- D016H)

This constant contains the address of the color register for play-field 0.

CRSINH 4

(-- 2F0H)

This constant contains the address of the cursor inhibition control register.

DZOOM-DL 4

(-- BE40)

This constant contains the starting address of the display list for the double-zoom mode.

M1-DL 4

(-- BCD0H)

This constant contains the starting address of the display list for BASIC mode 1.

M12-SCR 4

(-- B910H)

This constant contains the starting address of the screen data RAM for BASIC mode 1.

MAIN-DL 4

(-- BD10H)

This constant contains the starting address of the display list for the main graphics screen.

MAIN-SCR 4

(-- 9380H)

This constant contains the starting address of the screen data RAM for the main graphics screen.

MENU-DL 4

(-- BD80H)

This constant contains the starting address of the display list for the main menu screen.

MENU-SCR 4

(-- A010H)

This constant contains the starting address of the screen data RAM for the main menu screen.

QZOOM-DL 4

(-- BEDOH)

This constant contains the starting address of the display list for the quadruple-zoom mode.

2. Variables and Arrays for FIGE

BRPX 12

(brush# -- X-reference)

This array contains the X-reference values for the five single-line brushes.

BRPY 11

(brush# -- Y-reference)

This array contains the Y-reference vlaues for the five single-line brushes.

BRUSH 11

(brush# -- address)

This array contains the starting addresses of the shape data for the five single-line brushes.

BSH-POOL 11

(-- address)

This variable defines the starting address of the brush shape data pool.

CMD-ATTR 10

(command# -- attribute)

This array contains the collection of attributes for the 25 commands.

CMD-CFA 10

(command# -- cfa)

This array contains the collection of CFAs for the 25 commands.

CMD-NAME 10

(command# --)

This array is a character string array which contains the collection of the command names for the 25 commands.

CMDVEC 4

(-- address)

This variable contains the CFA of the current command.

CMENU 10

(-- address)

This variable and its following 158 bytes contain the shape data for the player which is used as the color menu.

COLNO 7

(-- address)

This variable is used as a counter during the execution of DLIRTN2 to count the number of times the display list interrupts has occurred after the recent vertical blank interrupt occured. It is also used as an index to the color pool COLRPOOL.

COLRPOOL 7

(-- address)

This variable and its following 15 bytes contain the color values which will be displayed in the color menu when the DLI occurs.

CURR-BRUSH 4

(-- address)

This variable contains the current brush number.

CURR-CMD 4

(-- address)

This variable containds the current command number.

CURR-ZOOM 4

(-- address)

This variable contains the current zoom mode number.

DBRPX 12

(brush# -- X-reference)

This array contains the X-reference values for the five double-line brushes.

DBRPY 12

(brush# -- Y-reference)

This array contains the Y-reference values for the five double-line brushes.

DBRUSH 11

(brush# -- address)

This array contains the starting addresses of the shape data for the five double-line brushes.

DISKVEC 39

(operation# -- cfa)

This array contains the CFAs of the disk operations.

EXITFLG 4

This variable contains a flag value which determines whether or not to exit FIGE.

HLCFA 18

(brush# --)

This array contains the CFAs of the words which draw horizontal lines for the five brushes.

HLVEC 4

(-- address)

This variable contains the CFA of the word which draws a horizontal line for the current brush.

MARKER 12

(-- address)

This variable and its following 21 bytes contain the shape data for a marker used to mark the current color and the current brush.

PLBTN 29

(-- address)

This variable contains the value which is previously read from the left-hand button of the KoalaPad.

PLCFA 18

(brush# -- cfa)

This array contains the CFA's of the words which plot points for the five brushes.

PLVEC 4

(-- address)

This variable contains the CFA of the word which plots points for the current brush.

VLCFA 18

(brush# -- cfa)

This array contains the CFA's of the words which draw vertical lines for the five brushes.

VLVEC 4

(-- address)

This variable contains the CFA's of the words which draws a vertical line for the current brush.

XYFLAG 4

(-- address)

This variable contains a flag value which determines to whether or not display the current cursor position in the message section.

3. High-level and Low-level Words for FIGE

!DZOOM-DL 6 H

(--)

This command creates a display list for the double-zoom mode.

!M1-DL 6 H

(--)

This command creates a display list with BASIC mode 1.

!M2-DL 6 H

(--)

This command creates a display list with BASIC mode 2.

!MAIN-DL 5 H

(--)

This command creates a display list for the main graphics screen.

!MENU-DL 5 H

(--)

This command creates a display list for the main menu screen.

!QZOOM-DL 6 H

(--)

This command creates a display list for the quadruple-zoom mode.

>CMD# 43 H

(X Y -- command# flag)

This command attempts to convert the given X-Y coordinates to a command number; if it can, the flag value will be set to 1.

>CXYR 26 H

(X-1 Y-1 X-2 Y-2 -- X-center Y-center radius)

This command converts the two given points to a center point and a radius of a circle.

CHECK-NAME 38 H

(-- address flag)

This command repeatedly queries the user for a filename until a new filename, with respect to the directory, is obtained. It then puts the address of the directory entry and a flag with value 1 on top of the parameter stack.

CLN 27 H

(X-1 Y-1 X-2 Y-2 --)

This word draws a line between the two given points. The breadth of the line depends on the shape of the current brush.

CLRM1 23 H

(--)

This command clears the whole BASIC mode 1 screen.

COLOR! 24 H

(color-value --)

This command stores the given color value in the current color register.

DELLINE 23 H

(line-1 line-2 --)

This command deletes the data between the two given lines in a BASIC mode 1 screen.

DISK 40 H

(--)

This command displays the disk operation menu and allows the user to select one of the disk operations.

DLIRTN1 7 L (ASM ONLY)

(--)

This word is a assembly language subroutine which serves as a display list interrupt service routine for the main graphics screen to change colors in the text window of that screen.

DLIRTN2 7 L (ASM ONLY)

(--)

This word is a assembly language subroutine which serves as a display list interrupt service routine for the main menu screen to change colors on that screen. The color menu is obtained with the aid of this service routine.

DRAW 35 H

(--)

This command draws points determined by the trace of the brush (or cursor) which is handled by the KoalaPad Touch Tablet.

DRAWLN 28 H

This command draws a line between the two points maintained in the FORTH variable, OLDCOL, OLDROW, NEWCOL and NEWROW. The breadth of this line depends on the shape of the current brush.

DZOOM 31 H

(--)

This command sets the required data for the double-zoom mode environment and turns on that mode.

ERASE 35 H

(--)

This command erases points determined by the trace of the brush (or cursor) which is handled by the KoalaPad Touch Tablet.

EXEC-CMD 45 H

(command# --)

This command executes the command with the given command number by means of vectored execution.

EXEC-ICMD 43 H

(X Y command# --)

This command executes the command with the given command number by means of vectored execution. X and Y are used during the execution of the given command.

EXIT 41 H

(--)

This command allows the user to exit FIGE. It displays a warning message to make sure that the current screen is saved before leaving, if that is desired.

GET-CMD 44 H

(-- command# flag)

This word allows the user to select one of the commands from the main menu by using the KoalaPad. If the selected command is an immediate command, GET-CMD executes it immediately; otherwise, the command number and a proper flag are put on top of the parameter stack.

GET-NAME 37 H

(-- address flag)

This command repeatedly queries the user for a filename and checks in the directory until the filename has been found. It then puts the address of the directory entry and a flag with value 1 on top of the parameter stack.

GETPOINT 21 H

(-- X Y flag)

This command repeatedly reads position values from the KoalaPad and moves the cursor until a proper button is pressed. If the read position values are valid, the read position values and a proper flag value are put on top of the parameter stack.

GETXY 21 H

(-- X Y)

This command reads position values from the KoalaPad and moves the cursor to that position.

GOODPOINT? 30 H

(-- flag)

This word examines the point maintained in the FORTH variable NEWCOL and NEWROW to see whether it is noise or not; if not, a flag with value 1 is returned.

HDWPTS 27 H

(inc1 inc2 difference end begin --)

This word applys Bresenham's Line Algorithm to draw a line by using the incremental values. The breadth of the line depends on the shape of the current brush.

HLN1 17 H

(X1 Y1 X2 Y2 --)

This word draws a horizontal line with breadth corresponding to brush 1.

HLN2 17 H

(X1 Y1 X2 Y2 --)

This word draws a horizontal line with breadth corresponding to brush 2.

HLN3 17 H

(X1 Y1 X2 Y2 --)

This word draws a horizontal line with breadth corresponding to brush 3.

HLN4 17 H

(X1 Y1 X2 Y2 --)

This word draws a horizontal line with breadth corresponding to brush 4.

INITENV 46 H

(-- flag)

This word displays the title page for FIGE, prompts the user to enter "GO" from a keyboard to load the main menu, and initializes the environment.

MARK-BRUSH 18 H

(--)

This word marks the box of the current brush with two bright horizontal lines.

MARK-COLOR 23 H

(--)

This word marks the box of the current color with two bright horizontal lines.

NCMD? 20 H

(command# --)

This word displays the corresponding command name with the given command number on the message section with a label "NEW:".

NZOOM 31 H

(--)

This command sets the required data for the normal-zoom mode environment and turns on that mode.

OCMD? 20 H

(command# --)

This word displays the corresponding command name with the given command number on the message section with a label "CMD:".

ONE-POINT 21 H

(-- X Y flag)

This command gets one cursor position by using the KoalaPad.

PADDRAW 30 H

(--)

This command allows the user to draw points determined by the trace of the brush by using the KoalaPad Touch Tablet.

PADPLOT 47 H

(--)

This command allows the user to start FIGE.

PLOT1 13 H

(X Y --)

This word draws points around the given point determined by the shape of brush 1.

PLOT2 13 H

(X Y --)

This word draws points around the given point determined by the shape of brush 2.

PLOT3 13 H

(X Y --)

This word draws points around the given point determined by the shape of brush 3.

PLOT4 13 H

(X Y --)

This word draws points around the given point determined by the shape of brush 4.

PRINTER 40 H

This word shows a message: "RESERVED FOR PRINTER".

QZOOM 32 H

(--)

This command sets the required data for the quadruple-zoom mode environment and turns on that mode.

RESETDLI 8 H

(--)

This command turns off the display list interrupt feature.

SET-BRUSH 19 H

(X Y --)

This command converts the given X-Y coordinates to a brush number and marks the corresponding box of the brush shape as the current brush.

SET-FIGS 19 H

(0 address X-reference Y-reference --)

This word assigns the given player shape data to all the four players.

SETCOLOR 25 H

(--)

This command displays a color menu which overlays the left-hand side of the main menu and allows the user to select a color for the current color.

SETDLI 8 H

(address --)

This command assigns an assembly language subroutine, whose entry point is at the given address, and the display list interrupt service routine is enabled.

SETLUMIN 23 H

(X Y --)

This command adjusts the luminance of the current color to either darker or brighter determined by the given X-Y coordinate.

SETPOINT 29 H

(player# -- X Y flag)

This word uses the given player as a cursor to select a point from the KoalaPad and maintains the selected point in a pair of proper FORTH variables for PADDRAW.

SETZOOM 24 H

(X Y --)

This command sets the current zoom mode to a zoom mode determined by the given X-Y coordinates.

SHOW-DMENU 36 H

(--)

This command displays the disk operation menu on the BASIC mode 1 screen.

SHOW-M1 36 H

(--)

This command displays the BASIC mode 1 screen.

SHOW-MENU 34 H

(--)

This command displays the main menu on the display.

SHOW-SCREEN 33 H

(--)

This command displays the main graphics screen on the display.

THREE-POINTS 22 H

(-- X-1 Y-1 X-2 Y-2 X-3 Y-3 flag)

This command gets three points from the KoalaPad by using three different players as the pointing cursors.

TWO-POINTS 22 H

(-- X-1 Y-1 X-2 Y-2 flag)

This command gets two points from the KoalaPad by using two different players as the pointing cursors.

UCOLOR 24 H

(X Y --)

This command converts the given X-Y coordinates to a color number and marks the corresponding box of color as the current color.

USEBRUSH 19 H

(brush# --)

This command assigns the brush shape of the given brush number to be the current cursor shape.

USECOLOR? 24 H

(X Y - flag)

This word checks to see if the given X-Y coordinates are located in the region of the usecolor command entry.

VBDRTN 8 L (ASM ONLY)

(--)

This word is an assembly language subroutine which is called by the vertical blank interrupt service routine to continue the user defined part of service. This word simply resets the color counter used by DLIRTN2 to 0.

VLN1 16 H

(X-1 Y-1 X-2 Y-2 --)

This word draws a vertical line between the two given points. The breadth of the line depends on the shape of brush 1.

VLN2 16 H

(X-1 Y-1 X-2 Y-2 --)

This word draws a vertical line between the two given points. The breadth of the line depends on the shape of brush 2.

VLN3 16 H

(X-1 Y-1 X-2 Y-2 --)

This word draws a vertical line between the two given points. The breadth of the line depends on the shape of brush 3.

VLN4 16 H

(X-1 Y-1 X-2 Y-2 --)

This word draws a vertical line between the two given points. The breadth of the line depends on the shape of brush 4.

XCIR 26 H

(X-center Y-center X Y --)

This command draws a circle with center at the first point and with radius determined by the distance between the first point and the second point.

XCLEAR 35 H

(--)

This command clears the whole graphics screen if the user's response is to do so.

XDEL 39 H

(--)

This command deletes the given filename from the file directory and reassigns the used data sectors to the free sector list.

XFCIR 26 H

(X-center Y-center X Y --)

This command draws a filled circle with center at the first point and with radius determined by the distance between the first point and the second point.

XGLOAD 39 H

(--)

This command reloads the screen data and/or the display list from the given file. The filename is obtained from the input stream.

XGSAVE 39 H

(--)

This command saves the display list and/or the screen data as a file on disk. The filename is obtained from the input stream.

XTEXT 31 H

(X Y --)

This command draws a text string, entered interactively by user, on the display starting at the given point.

XY? 20 H

(X Y --)

This word displays the given X-Y coordinates in the message section with the label: "X,Y", if the value of XYFLAG is 1.

ZOOM 32 H

(--)

This command sets up the environment for either normal-zoom mode, double-zoom mode or quadruple-zoom mode determined by the current zoom mode number.

ZOOM? 20 H

(zoom# --)

This word displays the name of the current zoom mode in the message section with a label "ZOOM:".

APPENDIX D: SOURCE LISTING FOR FIGE

SCR #3

```
( GRAPHICS -- DEMO )

HEX
GRAPHICS DEFINITIONS
VOCABULARY DEMO
DEMO DEFINITIONS








-->
```

SCR #4

```
( GRAPHICS -- CONSTANTS )

BD10 CONSTANT MAIN-DL   9380 CONSTANT MAIN-SCR
BD80 CONSTANT MENU-DL   A010 CONSTANT MENU-SCR
BCD0 CONSTANT M1-DL     BCF0 CONSTANT M2-DL
BE40 CONSTANT DZOOM-DL  BED0 CONSTANT QZOOM-DL
B910 CONSTANT M12-SCR
2F0  CONSTANT CRSINH
0 VARIABLE CURR-CMD   0 VARIABLE CURR-ZOOM
0 VARIABLE XYFLAG     0 VARIABLE PLVEC
0 VARIABLE HLVEC      0 VARIABLE VLVEC
0 VARIABLE CMDVEC     0 VARIABLE CURR-BRUSH
0 VARIABLE EXITFLG

-->
```

```
SCR #5
( GRAPHICS -- DLS )

: !MENU-DL  ( -- )
    HERE >R MENU-DL DP ! 4830 , BF40 , B008 , 4E C,
    A010 , A 0 DO 9 0 DO 0E C, LOOP 8E C, LOOP
    4E0E , B000 , 7 0 DO 0E C, LOOP
    5 0 DO 8E C, 9 0 DO 0E C, LOOP LOOP
    42B0 , BF60 , 0202 , 4102 , BD80 , R> DP !
;
: !MAIN-DL  ( -- )
    MAIN-DL BEGIN-DL
    MAIN-SCR D 50 MODE-LINE
    BF60 2 4 MODE-LINE END-DL
    8D MAIN-DL 54 + C!
;
-->

SCR #6
( GRAPHICS -- DLS )
: !DZOOM-DL  ( -- )
    DZOOM-DL MAIN-SCR A E 28 !ZOOM-DLS
    DA DZOOM-DL 78 + C!
;
: !QZOOM-DL  ( -- )
    QZOOM-DL MAIN-SCR 8 E 14 !ZOOM-DLS
    D8 QZOOM-DL 3C + C!
;
: !M1-DL  ( -- )
    M1-DL BEGIN-DL M12-SCR 6 18 MODE-LINE END-DL
;
: !M2-DL  ( -- )
    M2-DL BEGIN-DL M12-SCR 7 C MODE-LINE END-DL
;
-->
```

```
SCR #7
( GRAPHICS -- DEMO )
D016 CONSTANT COLPFO 0 VARIABLE COLNO
1402 VARIABLE COLRPOOL 3424 , 5444 ,
  7464 , 9484 , B4A4 , D4C4 , F4E4 , 92 C,
CODE DLIRTN1  ( -- )
  PHA, CA # LDA, D017 STA,
  92 # LDA, D018 STA, PLA, RTI,
CODE DLIRTN2  ( -- )
  PHA, TXA, PHA, COLNO LDX, COLRPOOL ,X LDA,
  D015 STA, D018 STA, 0 # LDA, D01A STA,
  28 # LDA, D016 STA, CA # LDA, D017 STA,
  INX, COLNO STX, PLA, TAX, PLA, RTI,
-->
```

```
SCR #8
( GRAPHICS -- DEMO )
CODE VBDRTN  ( -- )
  0 # LDA, COLNO STA, VBDEXIT JMP,
: SETDLI  ( ADDR -- )
    SETDLIV ' VBDRTN SETVBD DLI-ON
;
: RESETDLI  DLI-OFF RESETVBD ;  ( -- )
-->
```

SCR #9

```
( GRAPHICS -- EDMO )
: INIT-SARRAY   ( BEGIN END -- )
    -FIND
    IF DROP CFA >R 1+ SWAP
      DO BL WORD HERE @ 1 =
        IF 1 BLK +! 0 IN ! BL WORD ENDIF
        HERE I J EXECUTE SARRAY!
      LOOP R> DROP
    ELSE 2DROP ." Undefined string array name "
    ENDIF
;
-->
```

SCR #10

```
( GRAPHICS -- EDMO )
18 A SARRAY CMD-NAME
0 18 INIT-SARRAY CMD-NAME
  POINT LINE DRAW ERASE TEXT BOX FBOX CIRCLE
  FCIRCLE FILL COPY MOVE CLEAR BCLEAR SETCOLOR
  DISK PRINTER SETZOOM SETZOOM SETCOLOR SETBRUSH
  LUMINACE EXIT GO USECOLOR
18 CARRAY CMD-ATTR -19 ALLOT
   0201 , 0000 , 0201 , 0202 , 0102 , 0303 , 0200 ,
   0010 , 1000 , 1010 , 1010 , 1010 , 10 C,
0 VARIABLE CMENU 9F ALLOT
CMENU A0 OVER C! 1+ A0 FF FILL
-->
```

SCR #11

```
( GRAPHICS -- DEMO )
4 ARRAY BRUSH  4 ARRAY DBRUSH
0 VARIABLE BSH-POOL -2 ALLOT
HERE 0 BRUSH ! 8002 , 80 C,
HERE 1 BRUSH ! E002 , E0 C,
HERE 2 BRUSH ! 8006 , 8080 , 8080 , 80 C,
HERE 3 BRUSH ! F802 , F8 C,
HERE 4 BRUSH ! 800A , 8080 , 8080 , 8080 , 8080 , 80 C,
HERE 0 DBRUSH ! 8004 , 8080 , 80 C,
HERE 1 DBRUSH ! E004 , E0E0 , E0 C,
HERE 2 DBRUSH ! 800C , 8080 , 8080 , 8080 ,
   8080 , 8080 , 80 C,
HERE 3 DBRUSH ! F804 , F8F8 , F8 C,
HERE 4 DBRUSH ! 8014 , 8080 , 8080 , 8080 , 8080 ,
   8080 , 8080 , 8080 , 8080 , 8080 , 80 C,
-->
```

SCR #12

```
( GRAPHICS -- DEMO )
0 VARIABLE MARKER -2 ALLOT
   14 C, FFFF , 0000 , 0000 , 0000 , 0000 ,
   0000 , 0000 , 0000 , 0000 , FFFF ,
4 CARRAY BRPX -5 ALLOT
   0 C, 1 C, 0 C, 2 C, 0 C,
4 CARRAY BRPY -5 ALLOT
   0 C, 0 C, 2 C, 0 C, 4 C,
4 CARRAY DBRPX -5 ALLOT
   0 C, 2 C, 0 C, 4 C, 0 C,
4 CARRAY DBRPY -5 ALLOT
   0 C, 0 C, 4 C, 0 C, 8 C,
-->
```

```
SCR #13
( GRAPHICS -- DEMO )
: PLOT1   ( X Y -- )
    OVER 1 - OVER PLOT OVER 1+ OVER PLOT PLOT
;
: PLOT2   ( X Y -- )
    OVER OVER 1 - PLOT OVER OVER 1+ PLOT PLOT
;
: PLOT3   ( X Y -- )
    OVER 3 + DUP 5 - DO I OVER PLOT LOOP PLOT
;
: PLOT4   ( X Y -- )
    SWAP OVER 3 + DUP 5 -
    DO DUP I PLOT LOOP SWAP PLOT
;

10 LOAD
```

```
SCR #14
( ERROR MESSAGES      fig-FORTH )
Stack empty
Dictionary full
Wrong address mode
Isn't unique
Value error
Disk address error
Stack full
Disk Error!
All's well
```

SCR #15

```
( ERROR MESSAGES     fig-FORTH )
Use only in Definitions
Execution only
Conditionals not paired
Definition not finished
In protected dictionary
Use only when loading
Off current screen
Declare VOCABULARY
Nothing is wrong at all
```

SCR #16

```
( GRAPHICS -- DEMO )
: VLN1 ( X1 Y1 X1 Y2 -- )
    OFFSET>T 0 0 OFFSET! >R >R >R
    1 - R> R> 1+ R> FBOX T>OFFSET
;
: VLN2  ( X1 Y1 X1 Y2 -- )
    >R >R 1 - R> R> 1+ VLINE
;
: VLN3  ( X1 Y1 X1 Y2 -- )
    OFFSET>T 0 0 OFFSET! >R >R >R
    2 - R> R> 2 + R> FBOX T>OFFSET
;
: VLN4  ( X1 Y1 X1 Y2 -- )
    >R >R 2 - R> R> 2 + VLINE
;
-->
```

SCR #17

```
( GRAPHICS -- DEMO )
: HLN1  ( X1 Y1 X2 Y1 -- )
    >R >R >R 1 - R> R> 1+ R> OFFSET>T
    0 0 OFFSET! FBOX T>OFFSET ;
: HLN2  ( X1 Y1 X2 Y1 -- )
    OFFSET>T 0 0 OFFSET! >R >R
    1 - R> R> 1+ FBOX T>OFFSET
;
: HLN3  ( X1 Y1 X2 Y1 -- )
    >R >R >R 2 - R> R> 2 + R> OFFSET>T
    0 0 OFFSET! FBOX T>OFFSET ;
: HLN4  ( X1 Y1 X2 Y1 -- )
    OFFSET>T 0 0 OFFSET! >R >R
    2 - R> R> 2 + FBOX T>OFFSET
;
-->
```

SCR #18

```
( GRAPHICS --DEMO )

4 ARRAY PLCFA -A ALLOT
  ' PLOT CFA , ' PLOT1 CFA , ' PLOT2 CFA ,
  ' PLOT3 CFA , ' PLOT4 CFA ,
4 ARRAY HLCFA -A ALLOT
  ' HLINE CFA , ' HLN1 CFA , ' HLN2 CFA ,
  ' HLN3  CFA , ' HLN4 CFA ,
4 ARRAY VLCFA -A ALLOT
  ' VLINE CFA , ' VLN1 CFA , ' VLN2 CFA ,
  ' VLN3  CFA , ' VLN4 CFA ,
: MARK-BRUSH  ( -- )
    CURR-PLAYER C@ 2 USEPLAYER CURR-BRUSH C@
    10 * 78 PLAYER USEPLAYER
;
-->
```

```
SCR #19
( GRAPHICS -- DEMO )
: SETBRUSH   ( X Y -- )
    DROP 5 - F / CURR-BRUSH C! MARK-BRUSH
;
: SETFIGS   ( 0 PDATA RPX RPY -- )
    4 0 DO 2OVER 2OVER I SETPFIG DROP
        LOOP 2DROP 2DROP
;
: USEBRUSH   ( BRUSH# -- )
    >R R PLCFA @ PLVEC ! R HLCFA @ HLVEC !
    R VLCFA @ VLVEC ! CURR-ZOOM C@ 2 =
    IF 0 R DBRUSH @ R DBRPX C@ R DBRPY C@
    ELSE 0 R BRUSH @ R BRPX C@ R BRPY C@
    ENDIF SETFIGS R> DROP
;
-->
```

```
SCR #20
( GRAPHICS -- DEMO )
: XY?   ( X Y -- )
    XYFLAG C@ IF 19 0 TXT-LOCATE 10 SPACES
    14 0 TXT-LOCATE ." X,Y :" SWAP . ." ," .
    ELSE 2DROP ENDIF
;
: OCMD?   ( CMD# -- )
    2 0 TXT-LOCATE ." CMD : " CMD-NAME TYPE ;
: NCMD?   ( CMD# -- )
    2 1 TXT-LOCATE ." NEW : " CMD-NAME TYPE ;
: ZOOM?   ( ZM# -- )
    14 0 TXT-LOCATE ." ZOOM : " DUP 0=
    IF ." Normal" DROP
    ELSE 1 = IF ." Double" ELSE ." Quat  " ENDIF
    ENDIF
;
```

```
SCR #21
( GRAPHICS -- DEMO )
: GETXY   ( -- X Y )
    PADXY >SCRXY 2DUP ZPLAYER
;
: GETPOINT   ( -- X Y 1 [ or LR 0 ] )
  BEGIN LBTN C@ UNTIL
  BEGIN GETXY 2DUP XY? RBTN C@
    IF LBTM C@ IF 2DROP 0 ELSE 2DUP BOUNDARY?
                     IF 1 1 ELSE 2DROP 0 0 1 ENDIF ENDIF
    ELSE 2DROP 1 0 1 ENDIF
  UNTIL
;
: ONE-POINT   ( -- X Y 1 [or LR 0] )
    0 USEPLAYER GETPOINT
;
-->
```

```
SCR #22
( GRAPHICS -- DEMO )
: TWO-POINTS   ( -- X1 Y1 X2 Y2 1 [or 0] )
    ONE-POINT 1 FIXFLAG C!
    IF 1 USEPLAYER GETPOINT
      IF 1 ELSE 0 2SWAP 2DROP ENDIF
    ELSE 0 ENDIF 0 FIXFLAG C!
;
: THREE-POINTS   ( -- X1 Y1 X2 Y2 X3 Y3 1 [or 0] )
    TWO-POINTS 1 FIXFLAG C!
    IF 2 USEPLAYER GETPOINT
      IF 1 ELSE >R 2DROP 2DROP R> 0 ENDIF
    ELSE 0 ENDIF 0 FIXFLAG C!
;
3 ARRAY GETPTVEC -6 ALLOT
  ' ONE-POINT CFA , ' TWO-POINTS CFA ,
  ' THREE-POINTS CFA ,   -->
```

SCR #23

```
( GRAPHICS -- DEMO )
: SETLUMIN   ( X Y -- )
    DROP CURR-COLOR C@ SWAP 78 >
    IF BRIGHTER ELSE DARKER ENDIF
;
: MARK-COLOR   ( -- )
    CURR-PLAYER C@ 1 USEPLAYER CURR-COLOR C@
    20 * 10 + -16 PLAYER USEPLAYER
;
: CLRM1  M12-SCR 1E0 0 FILL ;   ( -- )
: DELLINE   ( L1 L2 -- )
    OVER - 1+ CURR-MODE C@ BY/LN C@ * 0
    ROT POSITION DROP SWAP 0 FILL
;
-->
```

SCR #24

```
( GRAPHICS -- DEMO )
: UCOLOR   ( X Y -- )
    DROP 10 - 20 / USECOLOR MARK-COLOR
;
: SETZOOM   ( X Y -- )
    DROP DUP 50 <
    IF DROP 0 ELSE 60 < IF 1 ELSE 2 ENDIF ENDIF
    DUP CURR-ZOOM C! ZOOM?
;
: USECOLOR?   ( X Y - F )
    0< OVER F > AND SWAP 90 < AND
;
: COLOR!   ( CVAL -- )
   CURR-COLOR C@ >CREG C!
;
-->
```

```
SCR #25
( GRAPHICS -- DEMO )
: SETCOLOR  ( -- )
  3 USEPLAYER 0 0 PLAYER 0 USEPLAYER
  BEGIN BEGIN LBTN C@ UNTIL 0 0
    BEGIN 2DROP GETXY LBTN C@ RBTN C@ AND NOT
    UNTIL RBTN C@
    IF 2DUP BOUNDARY?
      IF OVER 20 < IF A / COLRPOOL + C@ COLOR! DROP 0
         ELSE DUP 78 > OVER 8C < AND
         IF SETLUMIN 0 ELSE 2DROP 1 ENDIF ENDIF
      ELSE 2DUP USECOLOR?
        IF UCOLOR 0 ELSE 2DROP 1 ENDIF
      ENDIF
    ELSE 2DROP 1 ENDIF
  UNTIL 0 3 HPOSP0 + C!
;  -->

SCR #26
( GRAPHICS -- DEMO )

: >CXYR  ( CX CY X Y -- CX CY R )
    2OVER ROT - ABS >R - ABS DUP DUP * R R * +
    SWAP R> + 0 SWAP DUP 1+ SWAP 2/
    DO DROP I 2DUP DUP * <
      IF LEAVE ENDIF
    LOOP SWAP DROP
;
: XFCIR  ( CX CY X Y -- )
    >CXYR FCIRCLE
;
: XCIR  ( CX CY X Y -- )
    >CXYR CIRCLE
;
-->
```

```
SCR #27
( GRAPHICS -- DRAWLINE )
: HDWPTS ( INC1 INC2 DEFF XEND X )
    >R >R 0 R> R>
    DO I XSIGN @ + >R >R DUP XYSIGN @ +- 0<
      IF >R OVER R> + R>
      ELSE OVER + R> YSIGN @ +
      ENDIF R> OVER VL @ IF SWAP ENDIF
      PLVEC @ EXECUTE
    XSIGN @ +LOOP 2DROP 2DROP
;
: CLN ( X1 Y1 X2 Y2 -- )
    2OVER 2OVER 2OVER OFFSET>T OFFSET! ROT - ABS
    >R - ABS R> < VL ! >TEMP HCALINC
    0 0 PLVEC @ EXECUTE HDWPTS T>OFFSET
;
-->
```

```
SCR #28
( GRAPHICS -- CLINE )
: DRAWLN ( -- )
    NEWCOL @ NEWROW @ OLDCOL @ OLDROW @ 2OVER
    2OVER 2OVER DIRECTION 0=
    IF DROP HLVEC @ EXECUTE
    ELSE 0= IF VLVEC @ EXECUTE
            ELSE CLN ENDIF
    ENDIF
    OLDROW ! OLDCOL !
;
-->
```

SCR #29

```
( GRAPHICS -- SETPOINT )
0 VARIABLE PLBTN
: SETPOINT   ( N -- X Y F  )
    LBTN C@ PLBTN C! >R
    I USEPLAYER 0 0
    BEGIN 2DROP GETXY 2DUP XY? 2DUP I
      IF NEWROW ! NEWCOL !
      ELSE OLDROW ! OLDCOL !
      ENDIF 2DUP BOUNDARY?
      IF LBTN C@ PLBTN C@ AND
      ELSE  1
      ENDIF RBTN C@ AND  0=
    UNTIL RBTN C@ LBTN C@ 0= AND R> DROP
;
-->
```

SCR #30

```
( GRAPHICS -- PADDRAW  1/29/85  )
: GOODPOINT?   (  -- F )
    NEWCOL @ OLDCOL @ - ABS 5 <
    NEWROW @ OLDROW @ - ABS 5 < AND
;
: PADDRAW  (  --  )
    PLAYER-ON BEGIN RBTN C@ SETPF
    WHILE 0 SETPOINT >R 2DROP R> PLAYER-CLEAR
      IF BEGIN LBTN C@ NOT
         WHILE 1 SETPOINT
           IF GOODPOINT?
             IF DRAWLN  ENDIF
           ENDIF 2DROP PLAYER-CLEAR
         REPEAT
      ENDIF
    REPEAT PLAYER-OFF ; -->
```

```
SCR #31
( GRAPHICS -- DEMO )
: XTEXT  BF88 50 0 FILL 2 0 TXT-LOCATE TEXT ;   ( -- )
: NZOOM   ( -- )
    !MAIN-DL SCREEN-OFF MAIN-DL SDLSTL ! SCREEN-ON
    0 ZFLAG C! D PMODE C! SINGLE-LINE
    SETPLAYERS 3 0 DO I SINGLE-SIZE LOOP
;
: DZOOM   ( -- )
    !DZOOM-DL DZOOM-DL ZDL-ADDR ! 1 ZFLAG C! A PMODE C!
    50 ZXHIG ! 28 ZYHIG ! 50 ZMLX ! 28 ZMLY !
    SETPLAYERS 2 CK/PX C! ZX0 @ ZY0 @ POSITION FLUSHDL
    DOUBLE-LINE  3 0 DO I DOUBLE-SIZE LOOP
;
-->

SCR #32
( GRAPHICS -- DEMO )
: QZOOM   ( -- )
    !QZOOM-DL QZOOM-DL ZDL-ADDR ! 1 ZFLAG C! 8 PMODE C!
    78 ZXHIG ! 3C ZYHIG ! 28 ZMLX ! 14 ZMLY !
    SETPLAYERS 4 CK/PX C! ZX0 @ ZY0 @ POSITION FLUSHDL
    DOUBLE-LINE 3 0 DO I QUAD-SIZE LOOP
;
: ZOOM   ( -- )
    CURR-ZOOM C@ DUP 0=
    IF NZOOM ELSE DUP 1 = IF DZOOM ELSE QZOOM ENDIF
    ENDIF CURR-CMD C@ DUP 2 = SWAP 3 = OR
    IF CURR-BRUSH C@ USEBRUSH DROP
    ELSE 2 = IF 0 PDATA1 3 6
             ELSE 0 PDATA0 3 3 ENDIF SETFIGS
    ENDIF PLAYER-ON
; -->
```

SCR #33

```
( GRAPHICS -- DEMO )

: SHOW-SCREEN   ( -- )
    !MAIN-DL BF60 A0 0 FILL D PX/BY C@ PXBY C!
    D CURR-MODE C! 0 0 9F 4F TWINDOW TW>W
    MAIN-SCR SCR-MEMORY ! BF60 TXT-MEMORY !
    ' DLIRTN1 SETDLI ZOOM 1 XYFLAG C!
    4 SCROLL-LINES C! CURR-CMD C@ OCMD?
;






-->
```

SCR #34

```
( GRAPHICS -- DEMO )
: SHOW-MENU   ( -- )
    !MENU-DL BF60 A0 0 FILL ' DLIRTN2 SETDLI
    SCREEN-OFF MENU-DL SDLSTL ! SCREEN-ON
    MENU-SCR SCR-MEMORY ! BF60 TXT-MEMORY !
    1 26F C! 0 0 9F 9F TWINDOW TW>W
    4 SCROLL-LINES C! E CURR-MODE C! E PMODE C!
    SETPLAYERS PLAYER-ON CMENU 0 0 3 SETPFIG
    3 QUAT-SIZE MARKER 0 0 2 SETPFIG
    2 DOUBLE-SIZE MARKER 0 0 1 SETPFIG 1 QUAT-SIZE
    MARK-BRUSH MARK-COLOR CURR-CMD C@ OCMD?
    CURR-ZOOM C@ ZOOM? E PX/BY C@ PXBY C!
    0 ZFLAG C! 0 XYFLAG C! 0 USEPLAYER
    CURR-COLOR C@ USECOLOR
;
-->
```

SCR #35

```
( GRAPHICS -- DEMO )
: DRAW  ( -- )
    SHOW-SCREEN PADDRAW
;
: ERASE  ( -- )
    CURR-COLOR C@ >R 0 USECOLOR DRAW R> USECOLOR
;
: XCLEAR  (  -- )
    SHOW-SCREEN 2 1 TXT-LOCATE
    ." Are you SURE to discard the screen?"
    2 2 TXT-LOCATE
    ." Left btm : YES    Right btm : NO"
    BEGIN LBTM C@ DUP NOT
      IF CLEAR ENDIF RBTM C@ AND NOT
    UNTIL
;   -->
```

SCR #36

```
( GRAPHICS -- DEMO )
: SHOW-M1  ( -- )  !M1-DL
    SCREEN-OFF M1-DL SDLSTL ! SCREEN-ON
    6 CURR-MODE C! 6 PMODE C! SETPLAYERS PLAYER-ON
    M12-SCR SCR-MEMORY ! M12-SCR TXT-MEMORY ! 0 XYFLAG
    C! 0 0 13 17 TWINDOW TW>W 18 SCROLL-LINES C!
    CURR-MODE C@ PX/BY C@ PXBY C! 0 ZFLAG C! CLRM1
;
: SHOW-DMENU  ( -- )  !M1-DL
    1 0 SCR-LOCATE ." disk operation menu"
    3 2 SCR-LOCATE ." > DIRECTORY"
    3 3 SCR-LOCATE ." > SAVE SCREEN"
    3 4 SCR-LOCATE ." > LOAD SCREEN" 3 5 SCR-LOCATE
    ." > DELETE FILE" 3 6 SCR-LOCATE ." > FORMAT DISK"
    3 7 SCR-LOCATE ." > return" 1 8 SCR-LOCATE
;
```

```
SCR #37
( GRAPHICS -- DEMO )
: GET-NAME   ( -- ADDR F )
  0 >R BEGIN ROW-SCR C@ 2 * 4 OVER + DELLINE
           R IF ." file NOT found" ENDIF CR
           ." File Name:(XXXXX)" E SPACES QUERY
           TIB @ IN @ + C@ 0=
           IF 0 1
           ELSE FIND-FN
             IF 1 1
             ELSE R> DROP 1 >R 2 ROW-SCR C@
               1 - SCR-LOCATE 0
             ENDIF
           ENDIF
        UNTIL R> DROP
;
-->
```

```
SCR #38
( GRAPHICS -- DEMO )
: CHECK-NAME   ( -- ADDR F )
  0 >R BEGIN ROW-SCR C@ 2 * 4 OVER + DELLINE
           R IF ." file NOT unique" ENDIF CR
           ." File Name:(XXXXX)" E SPACES QUERY
           TIB @ IN @ + C@ 0=
           IF 0 1
           ELSE FIND-FN
             IF R> 2DROP 1 >R 2 ROW-SCR C@
               1 - SCR-LOCATE 0
             ELSE 0 1 1
             ENDIF
           ENDIF
        UNTIL R> DROP
;
-->
```

```
SCR #39
( GRAPHICS -- DEMO )
: XGSAVE   ( -- )
    CHECK-NAME IF 2 2 TXT-LOCATE 0 DLFLAG C! 1 DDFLAG
      C! CURR-ZOOM C@ 0 CURR-ZOOM C! SHOW-SCREEN
      ." <SCREEN SAVING>" 0 (GSAVE) CURR-ZOOM C! ENDIF
;
: XGLOAD   ( -- )
    GET-NAME CR IF ." <SCREEN LOADING>" 1 (GLOAD) ENDIF
;
: XDEL   ( -- )
    GET-NAME CR IF ." <FILE DELETING>" 1 (DELETE) ENDIF
;
4 ARRAY DISKVEC -A ALLOT
  ' DIR CFA , ' XGSAVE CFA , ' XGLOAD CFA ,
  ' XDEL CFA , ' FORMAT CFA ,
-->
```

```
SCR #40
( GRAPHICS -- DEMO )
: DISK   ( -- )
  BEGIN SHOW-M1 SHOW-DMENU
    BEGIN ONE-POINT
      IF SWAP DROP DUP 3 > OVER 10 < AND
        IF 4 - 2 / DUP 5 =
          IF DROP 1 1 ELSE DISKVEC @ EXECUTE 0 1 ENDIF
        ELSE DROP 0 ENDIF
      ELSE 1 1 ENDIF
    UNTIL
  UNTIL
;
: PRINTER   ( -- )
    2 2 TXT-LOCATE ." RESERVED FOR PRINTER"
;
-->
```

```
SCR #41
( GRAPHICS -- DEMO )
: EXIT  ( -- )
  BEGIN SHOW-M1 1 0 SCR-LOCATE ." it is time to say"
   3 1 SCR-LOCATE ." GOOD BYE" 1 3 SCR-LOCATE
   ." but be SURE to save" 3 4 SCR-LOCATE
   ." your NICE WORK!" 3 6 SCR-LOCATE ." . SAVE SCREEN"
   3 7 SCR-LOCATE ." . BACK TO SYSTEM"
   3 8 SCR-LOCATE ." . BYE-BYE" ONE-POINT
   IF SWAP DROP DUP B > OVER 12 < AND
     IF C - 2 / DUP
       IF 2 = IF 1 EXITFLG C! ENDIF 1
       ELSE DROP 1 9 SCR-LOCATE XGSAVE 0 ENDIF
     ELSE DROP 0 ENDIF
   ENDIF
  UNTIL
;  -->
```

```
SCR #42
( GRAPHICS -- DEMO )
18 ARRAY CMD-CFA -32 ALLOT
   ' PLOT CFA , ' LINE CFA , ' DRAW CFA ,
   ' ERASE CFA , ' XTEXT CFA , ' BOX CFA ,
   ' FBOX CFA , ' XCIR CFA , ' XFCIR CFA ,
   ' BFILL CFA , ' RCOPY CFA , ' RMOVE CFA ,
   ' XCLEAR CFA , ' BCLEAR CFA , ' SETCOLOR CFA ,
   ' DISK CFA , ' PRINTER CFA , ' SETZOOM CFA ,
   ' SETZOOM CFA , ' SETCOLOR CFA , ' SETBRUSH CFA ,
   ' SETLUMIN CFA , ' EXIT CFA , 0 ,
   ' UCOLOR CFA ,




-->
```

SCR #43

```
( GRAPHICS -- DEMO )
: >CMD#   ( X Y -- CMD# 1 or 0 )
    2DUP BOUNDARY?
    IF DUP 78 <
      IF 1E / 5 * SWAP 5 - 1E / +
      ELSE 78 - 14 / 2 * SWAP 50 / + 14 +
      ENDIF 1
    ELSE USECOLOR? IF 18 1 ELSE 0 ENDIF
    ENDIF
;
: EXEC-ICMD   ( X Y CMD# -- )
    DUP E = OVER 13 = OR OVER 16 = OR
    IF SWAP DROP SWAP DROP ENDIF
    CMD-CFA @ EXECUTE
;
-->
```

SCR #44

```
( GRAPHICS -- DEMO )
: GET-CMD   ( -- CMD# 1 or 0 )
   BEGIN
    BEGIN GETXY 2DUP >CMD#
     IF DUP NCMD? LBTN C@ IF 2DROP DROP 0 ELSE 1 ENDIF
     ELSE 2DROP 0 ENDIF
    UNTIL DUP 16 = OVER 17 = OR
    IF 16 = IF EXIT 2DROP 0
             ELSE 2DUP CURR-CMD C@ 1 ENDIF 1
    ELSE DUP CMD-ATTR C@ 10 =
     IF EXEC-ICMD
     ELSE DUP CURR-CMD C! OCMD? 2DROP
     ENDIF 0
    ENDIF
   UNTIL RESETDLI
;  -->
```

SCR #45

```
( GRAPHICS -- DEMO )
: EXEC-CMD  ( CMD# -- )
    DUP CMD-CFA @ >R CMD-ATTR C@ DUP
    IF SHOW-SCREEN
      BEGIN DUP GETPTVEC @ EXECUTE
        IF R EXECUTE 0 ENDIF PMBASE @ 800 0 FILL
      UNTIL RESETDLI
    ELSE R EXECUTE
    ENDIF R> 2DROP
;
-->
```

SCR #46

```
( GRAPHICS -- DEMO )
: INITENV  ( -- F )
    CA28 2C4 ! 4694 2C6 ! 0 2C8 C! SHOW-M1
    BEGIN CLRM1 13 RMGN C!
      5 0 SCR-LOCATE ." WELCOME TO" 7 1 SCR-LOCATE
      ." PADPLOT" 9 4 SCR-LOCATE ." BY" 4 5 SCR-LOCATE
      ." JER-MING LEE" 8 6 SCR-LOCATE ." 1985"
      0 7 SCR-LOCATE 14 0 DO ." _" LOOP 0 F 13 17
      WINDOW 2 8 SCR-LOCATE ." Insert Screen disk"
      ." Type 'GO' to start" QUERY TIB @ IN @ + DUP @
      4F47 = IF DROP CR ." <WAIT>" GLOAD 0 CURR-ZOOM C!
        0 CURR-BRUSH C! 0 CURR-CMD C! 1 CRSINH C! 1 1
      ELSE C@ IF 0 ELSE 0 1 ENDIF ENDIF
    UNTIL 27 RMGN C! MAIN-SCR C80 0 FILL 1 USECOLOR
;
-->
```

SCR #47

```
( GRAPHICS -- DEMO )

: PADPLOT  ( -- )
    INITENV 0 EXITFLG C!
    IF BEGIN SHOW-MENU GET-CMD
           IF EXEC-CMD ENDIF
           EXITFLG C@
         UNTIL
         MODE0 0 CRSINH C! 2 0 TXT-LOCATE
         ." EDN OF PADPLOT" CR PLAYER-OFF
    ENDIF
;

FORTH DEFINITIONS
DECIMAL
;S
```

REFERENCES

1. Foley, J. D. and A. V. Dam, Fundamentals of Interactive Graphics, Addison-Wesley Publishing Company, pp. 3.

2. Tenney, G. S., "TURTLETALK: A LOGO-like turtle graphics environment," 1981 FORML Proceedings, vol. 2, p. 521.

3. Miller, D., "The DATAHANDLER: A Simple, High-Speed Database Application in MMSFORTH," 1982 Rochester Forth Conference on Data Bases and Process Control, p. 39.

4. Bowhill, S. A. and F. Keasler, "FORTH System for Radar Control and Data Aquisition Using a Microcomputer," 1982 Rochester Forth Conference on Data Base and Process Control, p. 313.

5. Foley, J. D. and A. V. Dam, Fundamentals of Interactive Graphics, Addison-Wesley Publishing Company, p. 432.

6. Foley, J. D. and A. V. Dam, Fundamentals of Interactive Graphics, Addison-Wesley Publishing Company, p. 433.

7. Foley, J. D. and A. V. Dam, Fundamentals of Interactive Graphics, Addison-Wesley Publishing Company, p. 442.

8. Foley, J. D. and A. V. Dam, Fundamentals of Interactive Graphics, Addison-Wesley Publishing Company, p. 445.

9. Foley, J. D. and A. V. Dam, Fundamentals of Interactive Graphics, Addison-Wesley Publishing Company, p. 442.

10. Foley, J. D. and A. V. Dam, Fundamentals of Interactive Graphics, Addison-Wesley Publishing Company, p. 448.

11. Foley, J. D. and A. V. Dam, Fundamentals of Interactive Graphics, Addison-Wesley Publishing Company, p. 450.

REFERENCES NOT CITED

1. Rogers, D. F. and J. A. Adams, Mathematical Elements for Computer Graphics, 1978.

2. Compute!'s Second Book of ATARI, Published by Compute! Books, 1982.

3. Chadwick, I., Mapping the ATARI, Compute! Publications, Inc, 1983.

4. Leventhal, L. A., 6502 Assembly Language Programming, Osborne/McGraw-Hill, 1979.

5. Brodie, L., Starting FORTH, Prentice-Hall, 1982.

6. Derick, M. and L. Barker, FORTH Encyclopedia, Mountain View Press, 1982.